FACULTEIT ECONOMIE EN
BEDRIJFSWETENSCHAPPEN

KU Leuven

CONTRIBUTIONS TO COMPLEX PROJECT AND MACHINE SCHEDULING PROBLEMS

Dissertation presented to obtain
the degree of Doctor of Philosophy
in Business Economics

by

Morteza DAVARI

Number 554 JAN 2017

Doctoral Committee

Advisor:	Prof. dr. Erik Demeulemeester KU Leuven
Members:	Prof. dr. Roel Leus KU Leuven
	Prof. dr. Stefan Creemers KU Leuven and IESEG School of Management
	Prof. dr. Mario Vanhoucke Universiteit Gent
	Prof. dr. Christoph Schwindt Clausthal University of Technology

Acknowledgments

Undertaking this PhD has been a truly life-changing experience for me and it would not have been possible without the support and guidance that I have received from many people.

First and foremost, I wish to express my gratitude to my supervisor, Erik Demeulemeester, whose expertise, understanding and patience have added considerably to my graduate experience. I greatly appreciate Erik's vast knowledge and wisdom (in many areas, specially in the field of project scheduling) and his continued guidance and support throughout my doctoral program. In my opinion, Erik is a tremendous mentor: while he normally had a clear and wise idea how things should be done, he often pushed me to rely on my own creativity to seek the right approach and whenever I thought I had reached a dead end, he wisely guided me to find a way out. I truly appreciate all those countless very long technical and sometimes personal discussions Erik and I have had during the past four years. I have to admit that it is still a mystery to me how Erik manages his time such that whenever I asked "do you have time for a (short) meeting?" he simply answered "sure, come in". I would like to thank Erik for trusting me and giving me this amazing opportunity to pursue a PhD under his supervision. It has been a pleasure to work with him and I would certainly like to continue our collaboration.

I would like to express my sincere thanks to Roel Leus, who is an internal member of my Doctoral Committee. My history with Roel goes back to when I was living and studying in Iran, at which time we collaborated on a parallel machine scheduling problem. Thanks to this collaboration, I decided to travel to this beautiful and calm country to continue my graduate studies. Since the very first days that I arrived to Leuven, he has offered me direction and technical and personal support, and became more of a mentor and friend rather than a professor, which has been highly valuable for me. Moreover, in the course of my PhD, Roel and I

have had very constructive discussions that greatly improved the quality of this dissertation. He has greatly contributed to some chapters of this dissertation and is a co-author of some of my publications. I thank him for the trust, the insightful discussions and the valuable advice he has provided during my stay in Leuven.

I am also indebted to the other internal member of my Doctoral Committee, Stefan Creemers, whose support has been greatly valuable. Apart from providing me with numerous remarks and many novel ideas, Stefan was also a huge source of motivation and energy during the last two years, which I truly appreciate. I would also like to extend my thanks to the external members of my Doctoral Committee, Mario Vanhoucke and Christoph Schwindt. They read each chapter of this dissertation thoroughly and provided me with detailed constructive comments that considerably improved the correctness and the quality of this dissertation. Mario questioned my research more from practical and managerial points of view whereas Christoph looked at my research more from a mathematical angle.

A very special thanks goes to my former promoter and teacher and my current friend, Mohammad Ranjbar, without whose motivation and encouragement I would not have considered to pursue a graduate career in Operations Research. Mohammad is one of the few people who have truly made a difference in my academic life. It was through his persistence, understanding and kindness that I completed my undergraduate degree and was encouraged to apply for graduate education abroad.

There are a number of people who were (in)directly involved in my dissertation. I would like to thank Fabrice Talla Nobibon, who collaborated in some chapters of this dissertation and was a great mentor, Jeroen Beliën, who supported me during the hardships and made me realize that nothing is impossible, and Patricio Lamas, with whom I shared research ideas and with whom I had many theoretical and technical discussions.

I have been privileged to have had the opportunity to work in a wonderful atmosphere created by wonderful colleagues. I would like to express my deep gratitude to all my colleagues and friends in the 4th and 5th floor who made this journey incredibly special. I thank them all for those interesting, funny and sometimes philosophical discussions we have had during the lunch and coffee breaks, and for those extra-curricular activities and events we participated in together: the soccer training sessions and soccer tournaments, the volleyball games, the bowling competitions, the team building activities, the housewarming parties and barbecues, the spontaneous travels to Pizzeria Mangia & Via, the occasional running trips, the

amazing board games, etc. My thanks go to all my colleagues: Alexander, Ali, Amir, Ann, Annette, Bart S., Bart V., Ben, Brecht, Bo, Carla, Catherine, Celien, Daniel, Dennis, Dries, Eugen, Fan, Gert, Guopeng, Guoxuan, Hamed, Ines, Inneke, Jianjiang, Joeri, Joren G., Joren M., Jorne, Kris, Luca, Maud, Mieke, Michael, Narges, Nico, Nishant, Philip, Pieter, Raisa, Robert, Roel V., Rosita, Ruben, Salim, Saeed, Sarah, Sebastian, Stef, Thomas, Tim, Valeria, Vikram, Ward, Xuejun, Yannick and Zhiqiang, and my special thanks go to those who became really good friends.

Additionally, I am really grateful for having so many Iranian friends in Leuven, and I really appreciate all the support I have received from them. My special thanks go to Ali, Amir Hossein, Mohammadreza, Reza, Salim and their wives/girlfriends for their continued support and all those game nights, parties and barbecues we have had during the last four years.

I would like to express again and again my eternal gratitude to my family, specially my parents. Without their support and guidance, I would not have become the person that I am today. A distance of six thousands kilometers does not mean much to them: they always worry about me, care about me, love me, encourage me and motivate me every single day, even if they can only be with me once a year.

Finally, my deepest appreciations go to my wife, Zinat, without whose love I could not have lived, let alone finished this dissertation. I cannot imagine how could I have brought this PhD-journey to a good end if she was not always there to calm me when I was frustrated, if she was not always there to motivate me when I was disappointed and if she was not always there to erase all those sad thoughts with her smile. I thank her for joining me in Leuven and all the other countless sacrifices she has made during past years.

Morteza Davari
Leuven, 23 January 2017

Abstract

Scheduling problems are, generally speaking, the problems of allocating scarce resources over time to perform a given set of jobs (activities). Due to its practical relevance, the field of scheduling has been one of the most studied fields since the early days of Management Science and Operation Research. Within the field of scheduling, many sub-fields have been studied during the last century, among which project scheduling and single machine scheduling have received special attention. In this dissertation, we mainly contribute in these two mentioned sub-fields. As such, this dissertation includes two parts: the first part includes contributions to complex project scheduling problems and the second part includes contributions to a single machine scheduling problem.

In the first part, we contribute (i) by introducing an integrated proactive and reactive resource-constrained project scheduling problem and by proposing several models that can solve the introduced problem, (ii) by presenting two very important classes of reactions and by investigating their relevance in optimal proactive and reactive policies and (iii) by developing a novel branch-and-bound algorithm that solves instances of chance-constrained resource-constrained project scheduling problem.

In the second part, we contribute (i) by introducing a generic single machine scheduling problem where the objective is to minimize the total weighted tardiness penalty and solutions are subject to both time-window constraints and precedence constraints, (ii) by proposing several lower bounding schemes for the introduced problem and by investigating the theoretical relevance and the complexities of the proposed lower bounds and (iii) by presenting two branching schemes and a number of dominance rules that combined construct two branch-and-bound algorithms that can solve instances of the introduced problem.

Contents

Part I

Complex project scheduling problems

Chapter 1

Introduction and literature review

Every new beginning comes from some other beginning's end.

- Seneca

Each year, numerous projects go over budget or drag on long after their planned completion times (Flyvbjerg et al., 2003). To finish a project on time and within budget, understanding the factors that influence the success of the project is indispensable. Pinto and Prescott (1990) consider the project schedule/plan as one of the ten most critical project success factors. They also indicate that critical project success factors often positively correlate with either the quality of the initial planning or the quality of the tactical operationalization. This practical importance of the planning in project management has attracted many authors to study project planning from both managerial and tactical points of view (Demeulemeester and Herroelen, 2002; Lock, 2007).

In the project management literature, several definitions of the term *project* have been provided. Klein (2000) gathered the most common elements of all these definitions to provide a general definition. In general, a project can be described as a one-time *activity* (which consists of a set of sub-activities) that has to be realized in a certain period of time using a limited number of resources to fulfill a certain objective (Klein,

2000). More specifically, in the project scheduling literature, authors often simply address the one-time activity as the project and refer to the sub-activities as activities (Demeulemeester and Herroelen, 2002). Therefore, in the remainder of this dissertation, similarly to many other papers in the project scheduling literature, the combination of the terms 'project' and 'activity' is used.

Project scheduling, as the intersection of scheduling theory and project management, determines when to start (and/or finish) activities in order to accomplish a predefined goal. For instance, the *resource-constrained project scheduling problem* (RCPSP) determines the start and completion times of each activity in order to minimize the total project duration (makespan). In the RCPSP, scheduling takes place while not only precedence constraints but also resource constraints are to be satisfied. To be completed, each activity requires a certain number of renewable resources employed for a certain (activity) duration.

The majority of the papers in the literature, both in machine scheduling and in project scheduling, deals with *deterministic* models. In the deterministic RCPSP, we assume that all information is known in advance and each activity can only be executed in a single way which is determined by a fixed duration and fixed resource requirements. Deterministic project scheduling, in particular the deterministic RCPSP, gained much attention from many authors over the last few decades (Artigues et al., 2008; Demeulemeester and Herroelen, 2002; Klein, 2000; Neumann et al., 2003).

In many, if not most, real-life projects, the deterministic RCPSP fails to generate good schedules in the presence of uncertainty. Assuming activity durations and/or resource availabilities to be deterministic is not realistic for most of the real-life situations. Activity duration uncertainty and resource availability uncertainty occur because of several reasons such as: (1) a lack of clear information and precise specification of what is required, (2) a lack of experience to accurately execute each particular activity, (3) a large number of influential activity success factors, (4) the complexity brought by activity interdependencies with other activities, (5) a limited and inaccurate analysis of the processes involved in the realization of an activity and (6) the possible occurrence of particular unexpected disrupting events or situations that might affect the activity duration or the resource availability (Chapman and Ward, 2007).

To construct a useful schedule in an uncertain environment, uncertainty should be incorporated into the scheduling problem. Ignoring uncertainty in scheduling may cost a lot of money and time due to unex-

pected schedule disruptions. An online survey by Hillson (2003) indicates that most of the sources of uncertainty are not unexpected, so they can be identified, assessed and managed proactively (Herroelen, 2005). In the literature, there exist two main methodologies dealing with uncertainty in RCPSP, namely stochastic (dynamic reactive) scheduling (Igelmund and Radermacher, 1983a,b; Stork, 2001) and proactive and reactive scheduling (Demeulemeester and Herroelen, 2011; Leus, 2003; Van de Vonder, 2006).

The literature on project scheduling with uncertainty is multidisciplinary and as such it can be divided by different solution methodologies (stochastic, proactive and reactive scheduling), by different types of uncertainty (activity duration uncertainty and resource uncertainty) or by different types of objective functions (time-dependent and cost-dependent objective functions[1]).

In this chapter, the state of the art with regards to RCPSPs is reviewed. We focus on proactive and reactive approaches for the RCPSP with duration uncertainty and time-dependent objective functions. Besides, we try to briefly review dynamic (completely reactive) approaches to solve the stochastic RCPSP and give a brief and general overview of the exact and heuristic approaches to solve the deterministic RCPSP.

In recent decades, several handbooks and book chapters have been published that focus on the deterministic and stochastic RCPSPs (Artigues et al., 2008; Demeulemeester and Herroelen, 2002, 2011; Klein, 2000; Neumann et al., 2003; Schwindt and Zimmermann, 2015a,b). Also, several review papers with different structures and different scopes have been published (Hartmann and Briskorn, 2010; Herroelen, 2005). Following the categorization by Herroelen (2005), we first briefly review the literature on the deterministic RCPSP (Section 1.1). Next, in the following sections, we survey RCPSP problems under uncertainty. In Section 1.2, different variants of the stochastic RCPSP are discussed. Then, proactive scheduling approaches to solve the non-deterministic RCPSP are surveyed in Section 1.3. Finally, reactive scheduling methods will be summarized in Section 1.4.

[1]There exists a danger of confusion between the term 'cost-dependent objective function' (which refers to a function that is financially of importance, *i.e.*, total weighted earliness and tardiness cost and the expected total net present value (De Reyck and Leus, 2008; Deblaere et al., 2011c)) and the term 'cost function' (which, in this review, is considered as an alternative term for the term 'objective function' and is not necessarily concerned financially).

1.1 The deterministic RCPSP

We define the deterministic RCPSP as an optimization problem and review the best known methodologies to solve this problem. The deterministic RCPSP is known to be NP-hard in the strong sense (Blazewicz et al., 1983). However, a number of relatively efficient mixed integer (linear) programming (MIP/MILP) formulations as well as other exact methods, such as branch-and-bound and branch-and-cut algorithms, have been presented to solve the problem.

The following data are given for each instance of the deterministic RCPSP:

- A set $N = \{0, 1, ..., n+1\}$ of activities where activities 0 and $n+1$ are the dummy start and dummy end activities. Each activity has an integer duration $p_i \geq 0, i = 1, \ldots, n$ with $p_0 = p_{n+1} = 0$.
- A set $\mathcal{R}$ of resource types. Each activity i requires r_{ik} units of resource type $k \in \mathcal{R}$ during its processing time. The resource availability of resource type k is denoted by R_k.
- A set E of precedence constraints among activities where $E \subseteq \{(i,j)|i,j \in N, i \neq j\}$. The pair $(i,j) \in E$ indicates that activity j cannot be started before activity i is completed. The transitive closure of E is denoted by $T(E)$ and its transitive reduction is denoted by $\bar{T}(E)$.

A schedule $\mathbf{s}$ is a vector of starting times $(s_0, ..., s_{n+1})$ such that s_i is the scheduled starting time of activity i. Note that s_0 represents the starting time of the project, therefore it equals zero. A conceptual formulation (RCPSP) can be written as:

$$\text{(RCPSP)} \quad \min s_{n+1}$$

subject to

$$s_j - s_i \geq p_i \qquad \forall (i,j) \in \bar{T}(E) \tag{1.1}$$

$$\sum_{i \in O_t} r_{ik} \leq R_k \qquad \forall k \in \mathcal{R}, t = 0, \ldots, \mathcal{T} \tag{1.2}$$

$$s_i \in \mathbb{Z}^+ \qquad \forall i \in N \tag{1.3}$$

where O_t denotes the set of activities in progress at time t and $\mathcal{T}$ is an upper bound for the project makespan (Artigues et al., 2008). In the

above formulation, the objective function minimizes the starting time of the dummy end activity (note that the starting time of the dummy end activity equals the end of the project). Constraints (1.1) enforce the precedence constraints among the activities and constraints (1.2) ensure that all resource availability constraints are satisfied.

Many MIP formulations for the deterministic RCPSP have been proposed. Klein (2000) gathers six different MIP formulations for the deterministic RCPSP. The first two formulations (which we refer to as F1 and F2) are the time-indexed formulations proposed by Pritsker et al. (1969) and Kaplan (1988) (the latter formulation was originally proposed for the RCPSP where preemption is allowed). The next two formulations are MIP formulations (which we refer to as F3 and F4) proposed by Klein (2000) where the way of defining variables is taken from the formulations of multi-level lotsizing problems. The last two formulations are the sequence-based formulation (which we refer to as F5) developed by Alvarez-Valdes and Tamarit (1993) and the blockwise time-indexed formulation (which we refer to as F6) introduced by Mingozzi et al. (1998). Klein (2000) tests the first four formulations mentioned above and claims that F1 and F3 are the most efficient ones (among the first four formulations) in terms of space consumption and speed. Artigues et al. (2008) presents a number of drawbacks for F5, among which the exponential number of constraints is the most crucial. Despite all the drawbacks, this formulation (F5) can become beneficial in finding efficient *linear programming* (LP) lower bounds (Demassey et al., 2005) and in the construction of robust schedules (Lamas and Demeulemeester, 2016). Other MIP/MILP formulations have also been introduced during the last few decades, for which we refer to Artigues et al. (2003) who propose a resource-flow formulation, Koné et al. (2011) who review many MILP formulations and propose event-based MILP formulations for the RCPSP and Schwindt and Zimmermann (2015a, Chapter 2), who present an updated version of the formulation proposed by Koné et al. (2011).

Plenty of exact algorithms for the RCPSP have been proposed during the last few decades. Branch-and-bound algorithms (Demeulemeester and Herroelen, 1992, 1997; Klein, 2000; Mingozzi et al., 1998; Sprecher, 2000) have been the most successful approaches to optimally solve small and medium-sized instances. Large-sized instances, however, have remained unsolved due to the high complexity of the problem and the very large size of the branch-and-bound tree.

Although many exact algorithms have been proposed to solve the RCPSP, none of them is efficient enough to solve large size instances.

The computational complexity of the RCPSP motivates many authors to propose heuristic or meta-heuristic approaches to solve the problem. Hartmann and Kolisch (2000) provide a classification and performance evaluation of existing heuristic approaches to solve the RCPSP. They introduce two categories of approaches: X-pass methods and meta-heuristics. In a more recent publication (Kolisch and Hartmann, 2006), they also investigate recent approaches and methods that do not belong to any category (other methods) and update the performance results of the recent approaches.

In the standard RCPSP problem, the objective function is to minimize the project makespan where preemption is not allowed, resource availabilities are constant through time and there exists no setup cost. The project manager, however, might be interested in other criteria and/or situations. Hartmann and Briskorn (2010) provide a complete overview of the most known variants and extensions of the standard deterministic RCPSP studied in the literature. They categorize the RCPSPs into five categories. In the first category, they review papers dealing with different activity concepts (such as preemption, setup times, multiple modes and trade-offs). In the second category, they survey papers dealing with different temporal constraints (such as minimal and maximal time lags, release dates and deadlines). In the third category, Hartmann and Briskorn (2010) review variants of the RCPSP with different resource types (for example non-renewable resources, partially renewable resources, cumulative resources, continuous resources and time varying resource capacities). In the fourth category, they summarize the literature on the RCPSP with various objective functions (for example time-based objectives, robustness objectives, cost-based objectives and multiple objectives) and in the last category, they list the papers published on multiple project RCPSP problems.

In the following, we provide a practical example for the RCPSP.

Example (A practical example)**.** *Consider a software development company in which the managers intend to develop an analytical software. The software (which can be defined as a project) consists of eight sub-routines (activities). Each sub-routine needs a certain number of programmers (resource requirements) to do the programming within a certain period of time (activity durations). Eight programmers work for the company (resource availability) and we assume that their performances are quite similar. Note that some sub-routines are called within other sub-routines and for debugging reasons, the programmers cannot start coding a sub-routine*

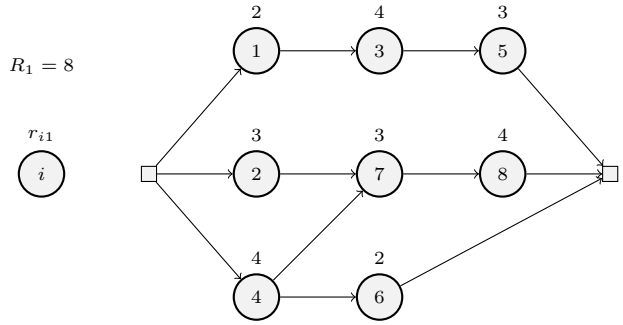


Figure 1.1: The precedence network for the example project.

before all necessary sub-routines are already properly working (precedence relations).

To find the earliest possible completion time of the project, we create a deterministic RCPSP instance. We consider an instance of eight non-dummy activities with one resource type of availability eight. The activity on node (AoN) representation of this instance is given in Figure 1.1. Given the deterministic vector of durations $\mathbf{p} = \{0, 2, 7, 4, 4, 8, 6, 4, 2, 0\}$*, an optimal solution* $\mathbf{s}^* = \{0, 0, 2, 4, 0, 7, 7, 9, 13, 15\}$ *for the RCPSP instance described above with makespan 15 has been computed by RESCON software*[2]*.*

1.2 RCPSP under uncertainty

As already motivated, during the project execution, project activity durations are subject to considerable uncertainty. Although in some cases uncertainties are negligible and/or avoidable, there still exist many real-world projects where uncertainties are important and inevitable. Considering deterministic activity durations is not realistic in such uncertain situations. Therefore activities together with their uncertain characteristics should be incorporated into the optimization problems. For activities

[2]RESCON is an educational software for the deterministic resource-constrained project scheduling problem (http://www.econ.kuleuven.be/rescon/) (Deblaere et al., 2011a). RESCON, which has been equipped with a version of the exact branch-and-bound algorithm of Demeulemeester and Herroelen (1992, 1997), can optimally solve the deterministic RCPSP.

with uncertain durations, one possibility is to represent activity durations by random variables.

The *stochastic RCPSP* (SRCPSP) is the integration of two well-known problems, namely the deterministic RCPSP and the stochastic project network problem. The SRCPSP is a variant of the standard RCPSP where the durations of the activities are random. The immediate questions are: "What is a solution?" and "What does the solution look like?" (Stork, 2001).

The SRCPSP is already known for many decades. Several papers have introduced such scheduling problems. The first mainstream international English-language publications studying the SRCPSP seem to be those written by Igelmund and Radermacher (1983a,b), although similar older researches were described by Kaerkes (1977) and Radermacher (1978). Within this mainstream, we also cite Möhring et al. (1985a) who provide an introduction for stochastic scheduling problems especially the SRCPSP and discuss several stochastic scheduling problems providing many examples and Möhring et al. (1984, 1985b) who focus more on the analytical aspects of these problems and provide analytical discussions and proofs. In the following subsections, we review other mainstream papers studying the SRCPSP and its related problems.

1.2.1 Dynamic project scheduling

Because durations are represented by random variables, the schedule can be constructed dynamically over time. This dynamic approach of scheduling is often called *completely reactive scheduling*.

One of the most attractive objective functions is to find a *policy* that minimizes the expected *project objective* value. A policy can be seen as a dynamic decision over time that decides which activity is being processed next according to the information (the realization of the activities that have been already scheduled and the probability characteristics of the activities waiting to be scheduled) given at the current time (Stork, 2001). The project objective is computed by a function that is assumed to be a *regular* function (*i.e.*, componentwise non-decreasing in the completion times of activities), for example the project makespan. We refer to Pinedo (2008) and Baker (1974) for a more precise definition and examples of regular (cost) functions.

Different classes of policies have been considered in the literature (Ashtiani et al., 2011; Möhring et al., 1984, 1985b; Stork, 2001), some of which will be briefly addressed in the following, namely the class of priority poli-

cies (also called resource-based policies (Ashtiani et al., 2011)), the class of early start policies, the class of pre-selective policies, the class of linear pre-selective policies, the class of activity-based policies (also referred to as job-based policies (Stork, 2001)) and the class of pre-processor policies. In the following, these classes of policies are discussed in more detail.

Priority policies The class of priority policies originated from the priority lists (rules). For each decision moment t, a priority policy schedules as many activities as possible (scheduling these activities must be resource and precedence feasible) based on a given order. Two drawbacks of this class are as follows: (1) using the policies belonging to this class, there exist instances for which none of the policies result in an optimal solution (Stork, 2001) and (2) the members of this class of policies suffer from Graham anomalies (Graham, 1969). One of the most important abnormalities is the possibility of increasing the project duration as a result of decreasing the duration of one or more activities. As already mentioned in the literature for many times, the policies belonging to this class are known to be "neither monotone nor continuous" (Möhring et al., 2000; Stork, 2000).

Early start policies The class of early start policies was introduced by Radermacher (1981). The idea is to convert a *directed acyclic graph* (dag) $G_0 = (N, E_0)$ to another dag $G_1 = (N, E_1), E_0 \subset E_1$ such that no *minimal forbidden set* exists in G_1. A set of activities is forbidden if it is an anti-chain in G_0 and the concurrent processing of its members results in a resource violation. A forbidden set is *minimal* if none of its subsets is a forbidden set. The early start policies, which are known to be continuous, monotonically increasing and also convex (Igelmund and Radermacher, 1983a; Stork, 2001), have been used several times in the literature (Artigues et al., 2013; Stork, 2001, 2000). Stork (2000) examines the early start policies in his branch-and-bound algorithm. Artigues et al. (2013) studies the maximum (absolute-)regret robust RCPSP where they try to find an early start policy that minimizes the maximum (absolute) regret over all scenarios.

Pre-selective policies and linear pre-selective policies The class of pre-selective policies is a generalization of the class of early start policies. The only difference is the addition of the *waiting activity*. Unlike early-start policies, in pre-selective policies, for each minimal forbidden

set FS, an activity (the waiting activity) is chosen and is delayed until at least one other activity in FS has been completed. The class of linear pre-selective policies is a subclass of the class of pre-selective policies. In linear pre-selective policies, a priority ordering of activities AL is given and the associated waiting activity for each minimal forbidden set is selected based on the given list. More specifically, the associated waiting activity of minimal forbidden set FS is considered to be activity $i \in FS$ that succeeds all activities $j \in FS \setminus \{i\}$ in AL. The class of pre-selective policies was introduced and examined by Igelmund and Radermacher (1983a,b). Stork (2001) follows Igelmund and Radermacher (1983a,b) and examines several classes of policies among which the class of pre-selective policies and its subclass, the class of linear pre-selective policies, are of the most importance. He also proposes a branch-and-bound algorithm to evaluate the performance of the mentioned classes of policies (Stork, 2000).

Activity-based policies The activity-based policies (or job-based policies (Stork, 2001)) are based on the same representation as the priority policies, but use a different procedure to construct the schedules. This procedure is called a stochastic serial *schedule generation scheme* (SGS), which is a mixture of a parallel and a serial SGS. Ballestín (2007) deploys three sampling procedures and three genetic algorithms to find an activity-based policy which minimizes the expected makespan. Activity-based policies have been used in the GRASP-heuristic algorithm proposed by Ballestín and Leus (2009) and in the artificial bee colony algorithm proposed by Tahooneh and Ziarati (2011) to minimize the expected makespan for the SRCPSP.

Pre-processor policies The class of pre-processor policies was introduced by Ashtiani et al. (2011). A pre-processor policy is defined by a set of activity pairs $X \subset N \times N \setminus T(E)$, with $G(N, E \cup X)$ being acyclic, and an activity list. The pairs in X, that impose extra finish-to-start precedence constraints, eliminate some of the minimal forbidden sets whereas the decisions associated with the remaining minimal forbidden sets are resolved dynamically during the project execution using the given activity list. Members of this class, which contain finish-to-start temporal arcs, combine the unconditional sequencing decisions of early start policies and the real-time dispatching features of resource-based policies (Ashtiani et al., 2011). Ashtiani et al. (2011) propose a two-phase local search procedure for their pre-processor policies. They also provide

improvements for early start policies and priority policies. Rostami et al. (2017) introduces a variant of the class of pre-processor policies whose members contain both finish-to-start and start-to-start arcs.

1.2.2 Proactive/reactive project scheduling

A very crucial disadvantage of dynamic (completely reactive) project scheduling is that no baseline schedule is constructed. A number of justifications for the necessity of a baseline schedule have been discussed in the literature (Herroelen, 2005): a baseline schedule (1) enables managers to allocate the resources to different activities, (2) provides overall information to set reliable due dates, (3) serves as a basis for planning external activities, such as preventive maintenance and order deliveries, (4) allows the monitoring and control of the project, (5) permits sharing the resources with other parties and (6) lets the manager arrange time windows with their subcontractors. For more details of the advantages of baseline schedules in machine and project scheduling problems we refer interested readers to Aytug et al. (2005), Demeulemeester and Herroelen (2011), Leus (2003) and Mehta and Uzsoy (1998).

An alternative solution methodology for solving the RCPSP with uncertainty is *proactive/reactive scheduling* which consists of two phases: proactive scheduling and reactive scheduling. Proactive scheduling is meant to construct a baseline schedule that is as robust (stable) as possible against duration uncertainty and/or resource uncertainty. While scheduling proactively, a reactive scheduling procedure may be needed when *conflicts*[3] occur. Note that proactive scheduling and reactive scheduling are not alternatives, but two complementary phases of planning (Chapman and Ward, 2007).

In proactive/reactive scheduling, the goal is to construct a solution that is as robust as possible against a certain type of uncertainty. The measures of robustness can be *single* or *composite*. *Quality robustness* and *solution robustness* are two well-known single robustness measures. In the following, we discuss the quality, solution and composite robustness measures. Other types of robustness and stability measures also exist in the literature, for which we refer to Chtourou and Haouari (2008), Mehta (1999), Sabuncuoglu and Goren (2009), Khemakhem and Chtourou (2013) and Schwindt (2005, Section 6.5).

[3]Conflicts are those disruptions that are not absorbed by the protected baseline.

Quality robustness Quality robustness is defined as a concept of protection against variations in the objective value. The most studied objective function is the project completion time (makespan). For this specific objective function, an example quality robustness measure is formulated as follows:

$$\pi(\mathtt{s}_{n+1} \leq \text{project's due date}), \tag{1.4}$$

where $\mathtt{s}_{n+1}$ denotes the realized starting time of the dummy end activity and operator $\pi(\cdot)$ represents the probability[4] of occurrence. The objective for this case is to maximize $\pi(\mathtt{s}_{n+1} \leq \text{project's due date})$ over all feasible schedules. Other examples of quality robustness measures are the minimization of the worst case makespan and the minimization of the maximum regret.

Solution robustness Solution robustness is defined as a concept of protection against variations in the starting times. Many authors have considered *starting time deviations*, which is the weighted sum of the differences between the realized starting times and the planned starting times, as their solution robust measure. For instance, Leus and Herroelen (2004) propose the following function as a measure of solution robustness:

$$\sum_{i \in N} w_i \mathrm{E}[|s_i - \mathtt{s}_i|] \tag{1.5}$$

where $\mathtt{s}_i$ denotes the realized starting time of activity i, s_i represents the planned starting time of activity i and $\mathrm{E}[\cdot]$ is the expectation operator. Notice that E refers to the set of precedence constraints among activities and must not be confused with the expectation operator. Also, w_i denotes the cost per unit time of the activity starting time deviation. The solution robust objective can be defined as the minimization of $\sum_{i \in N} w_i \mathrm{E}[|s_i - \mathtt{s}_i|]$ over all feasible schedules. Other examples of solution robustness measures are the minimization of the total weighted earliness/tardiness, the minimization of the total expected square deviations and the minimization of the total variance of completion times.

Composite robustness Any combination of the above single robustness measures can be considered a composite measure. Van de Vonder

[4] All probability operators in this dissertation adhere to the Kolmogorov's Axioms.

Method	Type of uncertainty	Papers
Proactive	Duration uncertainty	(Abbasi et al., 2006; Al-Fawzan and Haouari, 2005; Bruni et al., 2011; Chtourou and Haouari, 2008; Fu et al., 2012; Herroelen and Leus, 2001, 2004a,b; Klimek and Łebkowski, 2009; Lamas and Demeulemeester, 2016; Leus and Herroelen, 2004, 2005; Van de Vonder et al., 2005, 2006, 2007b, 2008)
	Resource uncertainty	(Lambrechts et al., 2007, 2008a,b, 2011)
Reactive	Duration uncertainty	(Van de Vonder et al., 2007a,b)
	Resource uncertainty	(Lambrechts et al., 2008a)

Table 1.1: The literature on the proactive and reactive RCPSP.

et al. (2008) introduce a composite measure of robustness as follows:

$$func(\pi(\mathtt{s}_{n+1} \leq \text{project's due date}), \sum_{i \in N} w_i \mathrm{E}[|s_i - \mathtt{s}_i|]), \qquad (1.6)$$

which is a bi-criteria objective where the first objective, which is the maximization of $\pi(\mathtt{s}_{n+1} \leq \text{project's due date})$, and the second objective, which is the minimization of $\sum_{i \in N} w_i \mathrm{E}[|s_i - \mathtt{s}_i|]$, are simultaneously considered for optimization.

Although a considerable number of papers deal with proactive project scheduling, the literature on reactive project scheduling remained rather scarce. Table 1.1 classifies some of the existing literature on the proactive and reactive RCPSP. In Section 1.3 and Section 1.4, we review the proactive and reactive approaches to solve the RCPSP under uncertainty,

respectively. Our classification is mainly elaborated based on duration uncertainty and resource availability uncertainty.

1.3 Proactive scheduling

Proactive scheduling, which is also known as predictive scheduling, has been considered by many authors when confronted with uncertainty in both machine and project scheduling problems (Aytug et al., 2005; Demeulemeester and Herroelen, 2011). Many authors have studied proactive (or predictive) approaches in single machine scheduling (for example Briskorn et al., 2011; Mehta, 1999), in parallel machine scheduling (for example Duenas and Petrovic, 2008), in job shop scheduling (for example Mehta and Uzsoy, 1998) and in resource-constrained project scheduling (for example Demeulemeester and Herroelen, 2011; Herroelen and Leus, 2004a,b; Leus and Herroelen, 2004).

Among all existing types of uncertainty, two types of uncertainty, namely duration uncertainty and resource availability uncertainty, have been focused on the most in the literature. In the following sections, we review the literature on each of these two types of uncertainty.

1.3.1 Uncertainty in activity durations

A vast number of papers in the literature, in both machine and project scheduling, are devoted to duration uncertainty. Duration uncertainty in project scheduling refers to the situation where the durations of the activities, for whatever reasons, are subject to uncertainty, which means that activities can last longer or shorter than what is planned.

A pioneer proactive scheduling method is the *critical chain scheduling and buffer management* (CC/BM) method which has been constructed based on the principles described by Goldratt (1997). By adding several types of buffers, namely feeding buffers[5], resource buffers[6] and project

[5]A feeding buffer is the safety time added when a non-critical chain activity joins a critical chain.

[6]Resource buffers are added to some activities on the critical chain to ensure the availability of the project resources. More precisely, a resource buffer is added to the end of an activity when its successor in the critical chain needs a renewable resource that is not used by the previous activity or needs more of at least one resource type than the preceding activity. This type of buffer has gained less attention from authors. Thus resource buffers are not discussed further in this review.

buffers[7], to the project (Goldratt, 1997), CC/BM creates proactive schedules to tackle uncertainty. In this context, many papers (for instance Herroelen and Leus, 2001; Herroelen et al., 2002; Van de Vonder et al., 2005) have been written to discuss the advantages and disadvantages of CC/BM.

The first footsteps of the research on solution proactivity in the RCPSP have been taken by Leus (2003). Leus and Herroelen (2005) show that scheduling with a single disruption and in the presence of precedence constraints is strongly NP-hard even for a single machine environment. Herroelen and Leus (2004a) propose exact methods (MIP formulation) to construct solution robust baseline schedules for the case in which only one disruption occurs. They also propose a number of heuristic procedures to obtain solution robust schedules, among which the *activity-dependent float factor* (ADFF) model is of most importance. They assume that the *railway scheduling policy*[8] is adopted, where the activities are not started earlier than their planned starting times even if the corresponding resource and precedence constraints are met. They additionally study how to allocate resources such that the protection of a given schedule against possible duration variability is maximized (Leus and Herroelen, 2004). They also present a resource allocation model that increases the stability of a given schedule.

Van de Vonder et al. (2005) seeks to understand the trade-off between quality and solution robustness. They run a number of simulations to compare the CC/BM approach, which provides quality robust schedules, with the ADFF model developed by Herroelen and Leus (2004a), which provides stable schedules. Both approaches add safety time (buffers) to the project in order to protect against unexpected duration variability. Despite similar protection methods (adding safety time), the two approaches have major differences and produce different schedules. The main differences between these two approaches are the way they distribute the safety time in the project and the type of robustness they provide. The CC/BM approach adds only project and feeding buffers whereas the ADFF model distributes the buffers throughout the project schedule. The former ap-

[7] A project buffer is the safety time added at the end of a project schedule to protect against aggressive duration estimates.

[8] The railway scheduling policy which is named by Van de Vonder et al. (2005) is one of the two best known execution scheduling policies. The alternative execution scheduling policy is called *roadrunner scheduling policy* where activities are started as soon as possible after the completion of all their predecessors provided that enough resources are available (Goldratt, 1997).

proach denies the importance of the intermediate milestones, follows the roadrunner mentality and therefore only tends to maintain the quality robustness of the schedule, whereas the latter one follows the railway scheduling approach and therefore aims at stability and solution robustness. Van de Vonder et al. (2005) conclude that the ADFF model not only constructs solution robust schedules, but also provides reasonable quality robustness while the CC/BM approach fails in providing solution robustness.

Several improved heuristic approaches have been studied in the literature (Van de Vonder et al., 2006, 2008). Van de Vonder et al. (2006) propose a *resource flow-dependent float factor* (RFDFF) heuristic to construct solution robust schedules. In fact, RFDFF is a modified version of ADFF, in which the resource flow is employed to avoid resource infeasibility. They set up a thorough analysis to investigate the impact of certain parameters (number of activities, weighting parameter, order strength (Demeulemeester et al., 2003), etc.) on the solution and quality robustness of the resulting schedule. In line with the research done by Van de Vonder et al. (2006), Van de Vonder et al. (2008) develop new heuristic procedures, among which the *virtual activity duration extension* (VADE) and the *starting time criticality* (STC) heuristics are the most efficient, and compare their performances with those already provided in the literature. According to their comparison, STC is ranked best in terms of the average computational speed of the procedure and the average solution robustness of the generated schedules.

Recently, robust methodologies have been provided by using chance-constrained programming (Birge and Louveaux, 2011). Bruni et al. (2011) employ a joint chance constraint to ensure a certain level of robustness in their proactive schedule. Their robust schedule consists of a vector of starting times and a vector of completion times. They use a *stochastic dynamic generation scheme* (SDGS) that heuristically determines the starting times based on the completion times of the predecessors and the resource availability. In SDGS a chance-constrained programming problem is iteratively solved to compute the completion times. Lamas and Demeulemeester (2016) introduce the chance-constrained RCPSP (CC-RCPSP) which is the chance-constrained version of F5, the sequence-based RCPSP formulation introduced by Alvarez-Valdes and Tamarit (1993). Lamas and Demeulemeester (2016) recognize three difficulties to solve their proposed CC-RCPSP: (1) the feasible region is not convex, (2) even a feasibility check for the joint chance constraint is difficult and (3) the number of minimal forbidden sets in F5 is exponential in the num-

ber of activities. They tackle the first two difficulties by sample average approximation and the third difficulty by a branch-and-cut algorithm.

Zheng et al. (2013) study a solution robust method for the resource-constrained multi-project scheduling problem. They propose a bi-criteria model consisting of two objectives: (1) maximizing a measure of solution robustness tailored for a multi-project environment and the minimization of the total makespan, and they develop an algorithm based on the multi-objective genetic algorithm known as NSGA-II.

Deblaere et al. (2011c) try to combine the notions of a proactive schedule and an execution policy (which maps a vector of activity durations to a vector of resource and precedence feasible starting times). They compute an optimal baseline schedule for every given execution policy. For every combination of a policy and a baseline schedule, a cost value is computed that is equivalent with the expected total weighted earliness and tardiness. Their objective is to minimize the total cost by choosing the best execution policy. Deblaere et al. (2011c) propose two heuristic algorithms, one initializes an execution policy and the other iteratively improves the quality of the initial policy in terms of its associated objective cost.

There are some research on the bi-criterion RCPSPs under uncertainty that seem to be interesting and related. Many heuristics have been proposed to solve the bi-criterion RCPSP under uncertainty. The most common bi-criterion objective function is the combination of the minimization of the makespan and the optimization of a predictive robustness measure. Examples are reviewed in the following. Al-Fawzan and Haouari (2005) propose a tabu search algorithm for the bi-criterion RCPSP. Their robustness measure is the total sum of the *free slacks*[9]. Abbasi et al. (2006) define another bi-criterion objective function, which is a linear function of the makespan and the total free slack, and solve it by a simulated annealing algorithm. Chtourou and Haouari (2008) define twelve different robustness measures and propose a two-stage robust schedule generation approach to solve the bi-criterion RCPSP.

Other related topics and approaches that have been studied during the last decade are reviewed in this paragraph. Klimek and Łebkowski (2009) study the robust buffer allocation for the RCPSP with predefined milestones. They introduce a buffer allocation algorithm that inserts unit buffers such that the robustness measure is maximized. Fu et al. (2012) study the RCPSP with minimum and maximum time lags and activity

[9]In Al-Fawzan and Haouari (2005), the free slack of activity i is the amount of time that activity i can slip without delaying the very next activity and while maintaining resource feasibility.

duration uncertainty (SRCPSP/max). They propose a local search algorithm to obtain a reasonably robust schedule for the SRCPSP/max.

1.3.2 Uncertainty in resource availability

Although most of the research on the RCPSP under uncertainty focuses on the activity duration uncertainty, resource availability uncertainty in the RCPSP has also gained much attention during the last decade. Resource availability uncertainty in project scheduling refers to the situation in which the resource availabilities are subject to uncertainties, which can be caused by resource failures or breakdowns.

There exist not much research on project scheduling with resource availability uncertainty. To the best of our knowledge, Lambrechts et al. (2007, 2008a,b, 2011) are the only authors who published on this topic. Lambrechts et al. (2008a) state that robust schedules can be constructed by inserting any combination of two types of buffers, namely time buffers and resource buffers, into any given initial schedule. They propose two approaches to construct the initial schedule and a number of heuristic algorithms to insert the resource and time buffers. They also provide three reactive approaches to respond to possible conflicts. In Lambrechts et al. (2008a) the objective function is the weighted sum of the starting time deviations.

In a parallel research, Lambrechts et al. (2008b) propose a tabu search procedure as a predictive/reactive project scheduling approach, in which they examine a free-slack based objective function to evaluate the performance. Although this tabu search cannot outperform the best approaches in Lambrechts et al. (2008a), one should not neglect the advantage of using the free-slack objective function: unlike the evaluation of the objective function in Lambrechts et al. (2008a), which is done by simulation, the free-slack objective function is computed in polynomial time. Lambrechts et al. (2011) also state that "time buffering based on simulation performs far better than surrogate objective functions, but the reader should keep the higher computational demands in mind". They propose time buffering procedures to construct robust schedules for the RCPSP with resource availability uncertainty. They introduce three surrogate measures of robustness and compare them with simulation-based time buffering and the STC procedure.

1.4 Reactive scheduling

As we already discussed in Section 1.3, proactive scheduling constructs a schedule that is protected against disruptions. When disruptions occur, this protection prevents the schedule from precedence and resource violations. However, it is still possible that the protection fails to absorb the disruption and therefore the schedule becomes infeasible or disturbed. Such disruptions are called conflicts. In such cases, a rescheduling approach, which is also known as reactive scheduling, may be used to make the schedule feasible or improve the schedule's robustness.

In reactive scheduling, providing a schedule that does not deviate much from the baseline schedule should be considered as one of the most important objectives. It is not always desirable, both from managerial and from planning points of view, that the rescheduled starting times of the activities differ a lot from the planned starting times of these activities. In a project with medium or high levels of uncertainty, because of high chances of disruptions, the planned starting times may inevitably differ to a large extent from the rescheduled starting times. In such cases, many important decisions such as the ones regarding personnel/resource schedule and delivery of raw materials need to be retaken, which often costs a lot.

As we already mentioned, the literature on reactive project scheduling is scarce. Being scarce, it has only been reviewed by Herroelen (2007) and Demeulemeester and Herroelen (2011). We classify the reactive project scheduling literature into two categories: reactive approaches for the RCPSP with activity duration uncertainty and reactive approaches for the RCPSP with resource availability uncertainty. In the following subsections, we review the few existing reactive approaches for the RCPSP with activity duration uncertainty and resource availability uncertainty.

1.4.1 Reactive RCPSP with activity duration uncertainty

Van de Vonder et al. (2007b) provide a classification of proactive and reactive procedures for the RCPSP with duration uncertainty. They propose four different reactive scheduling methods, namely complete rescheduling, an early-start policy after fixing resource flows, activity-based priority rules and minimizing earliness and tardiness costs. In the following, we discuss these four methods in more detail.

The first proposed reactive scheduling method is complete rescheduling. In complete rescheduling, Van de Vonder et al. (2007b) use the same procedure as used for constructing the baseline schedule with the only difference that the already scheduled activities are eliminated from the problem. There are two main drawbacks of this method: (1) the procedure requires high computational efforts and (2) the *projected schedule*[10] may deviate a lot from the baseline schedule.

The second proposed reactive scheduling method is to apply an early-start policy while maintaining the resource flow. As maintaining the resource flow from the baseline schedule certainly reduces the flexibility of the rescheduling procedure, it results in much faster computations and more robust schedules.

The third proposed reactive scheduling method relies on activity-based priority rules. The idea is to use a schedule generation scheme (SGS) to obtain a projected schedule from a priority list. Van de Vonder et al. (2007b) consider the *earliest baseline activity starting time* (EBST) as the priority list and the SGS procedure by Stork (2001). When a conflict occurs, the unrealized activities are started as soon as possible after the current time (current decision point).

The last reactive scheduling method proposed by Van de Vonder et al. (2007b) is to solve the *resource-constrained weighted earliness tardiness project scheduling problem* (RCPSPWET). The due dates in the RCPSPWET are set to the starting times of the activities in the baseline schedule and the earliness and tardiness penalties are set to provide the stability of the outcome schedule. To solve the RCPSPWET, they use a customized version of the exact procedure proposed by Vanhoucke et al. (2001). Although the procedure is computationally expensive, the results show that it reasonably preserves the stability. However, the makespan deviation might become large since the project may be delayed due to another conflict which occurs later during the project scheduling.

Van de Vonder et al. (2007a) propose three advanced heuristic reactive scheduling procedures, namely basic sampling, time-window sampling and a heuristic *weighted earliness tardiness* (WET) procedure. Their objective is to find a projected schedule with the smallest total weighted deviation from the baseline schedule. In the following, we discuss these three procedures in more detail.

[10]The projected schedule is the schedule that results when reactive scheduling is applied.

Sampling approach In the basic sampling approach, at each decision point t, 28 different schedules $\mathbf{s}^{l,k}$ are constructed by using four schedule generation schemes ($k = 1, ..., 4$) and seven priority lists ($l = 1, ..., 7$), the best schedule $\mathbf{s}^t$ with the smallest deviation from the predictive schedule is selected and all the activities with $s_i^t = t$ in $\mathbf{s}^t$ are set to start at time t. They use EBST, LST (latest starting time), LW (largest activity weight), LAN (lowest activity number) and RND (random) as static priority lists and EPST (earliest projected starting time) and MC (minimal cost) as dynamic priority lists. The four schedule generation schemes are the robust parallel SGS, the robust serial SGS, the parallel SGS and the serial SGS (Van de Vonder et al., 2007a).

Time-window sampling approach A drawback of the basic sampling approach is that the selection of a projected schedule depends on the starting times of all unscheduled activities including those that will start very late after the current time t. To overcome this drawback, they introduce the time-window sampling approach. In the time-window sampling approach the SGSs only define the starting times of the activities that start within a time-window $[t, t + \Delta]$. Van de Vonder et al. (2007a) show that the time-window sampling approach constructs more stable schedules than those constructed by the basic sampling approach[11], although it is computationally a bit more expensive than the basic sampling approach.

Heuristic WET procedure Van de Vonder et al. (2007a) also propose a heuristic weighted earliness and tardiness approach to construct schedules with high stability. In fact, in each decision point t the reactive scheduling step can be seen as a RCPSPWET. Van de Vonder et al. (2007a) argue that the existing exact approaches (Kéri and Kis, 2006; Vanhoucke et al., 2001) to solve the RCPSPWET are too slow for their purpose, so they use an adopted version of the population-based iterated local search algorithm proposed by Ballestín and Trautmann (2008). Van de Vonder et al. (2007a) depict that this heuristic approach constructs more robust solutions than those constructed by the basic or time-window sampling approaches. However, the heuristic WET procedure turns out to be much more time consuming than the two mentioned alternatives.

[11] Note that the robustness measure for this comparison is the total weighted starting times deviation from the baseline schedule. The result of a comparison with another robustness measure may be different.

1.4.2 Reactive RCPSP with resource uncertainty

To the best of our knowledge, there exists only one paper dealing with reactive scheduling procedures for the RCPSP with resource availability uncertainty (Lambrechts et al., 2008a). Lambrechts et al. (2008a) propose two different reactive approaches to deal with resource conflicts: list scheduling and a tabu search based heuristic.

List scheduling List scheduling is the first reactive scheduling approach proposed by Lambrechts et al. (2008a). The idea is to regenerate the schedule using an SGS and a priority list L. Lambrechts et al. (2008a) develop a modified version of the serial SGS (MS-SGS). At the time of disruption, the activities in progress are either rescheduled or left unchanged. The MS-SGS first tries to maintain the baseline starting times for these activities. If maintaining starting times is not possible, MS-SGS restarts these activities and reschedules them, together with other unprocessed activities, as soon as possible based on the given priority list. Two types of priority list are tested: a random priority list and a scheduled order list. The random priority list is a random list of activities and does not use any information from the baseline schedule. On the contrary, the scheduled order list is generated from the baseline schedule and provides stability. Lambrechts et al. (2008a) consider the result of the random priority list as a benchmark. The average stability provided by using the scheduled order list is far better than that provided by using the random priority list.

Tabu search improvement approach The solutions obtained by applying list scheduling can be improved using a tabu search improvement algorithm. In this algorithm, a search procedure iteratively exchanges the position of two adjacent activities in the given priority list while a tabu list prevents the recreation of the most recent moves. Using tabu search surely improves the stability of the projected solution, though it is computationally more expensive. So, in some situations where many immediate and fast reactions are necessary, tabu search might be less attractive than the simple list scheduling approach.

1.4.3 Other notable reactive approaches in project scheduling

Although our main focus is on reactive scheduling approaches for the RCPSP with activity duration or resource availability uncertainty, some notable borderline approaches are also discussed here.

Zhu et al. (2005) propose a general class of reactive scheduling problems and introduce a recovery problem. In their recovery problem, they consider six different types of uncertainty (disruptions) that are listed as follows: (1) a *new activity disruption* occurs when a set of new activities and their corresponding precedence constraints are added to the project, (2) a *precedence disruption* occurs when a number of precedence relations are removed from or added to the project, (3) an *activity duration disruption* can occur when the duration of an activity is subject to uncertainty, which is equivalent with the duration uncertainty focused in this thesis and discussed in Section 1.4.1, (4) an *activity resource disruption* occurs when an activity turns out to need more resources during execution than planned, (5) a *resource disruption* occurs when a resource unit breaks, which is equivalent with the resource availability uncertainty that is the focus of this thesis in Section 1.4.2, and (6) a *milestone disruption* occurs when the target time of a milestone moves. Zhu et al. (2005) also provide a very general MILP model for their proposed recovery scheduling problem and discuss it for a number of special cases, namely the resource unconstrained case, the single mode case, the multi-mode case, the case with one renewable resource and the case with one non-renewable resource. The recovery objective function for this general model considers the initial plan, the deviation from the initial plan and the cost to return back on track. They present a hybrid *mixed integer programming constraint propagation* (MIP/CP) solution approach to solve their MILP model.

Kuster et al. (2009) introduce a resource-constrained project scheduling problem, called x-RCPSP, and propose a disruption management approach for the same problem under uncertainty. x-RCPSP is a variant of RCPSP where a set of alternative activities are also included in the project where activities can be activated or deactivated dynamically. Each alternative activity possesses its individual sets of predecessors and successors. Thus, activating or deactivating an activity might have an impact on the activation or deactivation of some other activities. This fact differentiates x-RCPSP from the multi-mode RCPSP. In a complementary research, Kuster et al. (2010) propose three local rescheduling approaches as disruption management approaches and compare them with the full

rescheduling and the *matchup scheduling*[12] (Bean et al., 1991) known from literature.

Deblaere et al. (2011b) propose a model for the reactive multi-mode RCPSP under uncertainty and provide a number of tree-based search approaches, namely regular branch-and-bound, iterative deepening and branch-and-bound with tabu search, to select the best reactive schedule. They consider both activity duration disruptions and resource disruptions. In their model, Deblaere et al. (2011b) assume that only one disruption occurs and in cases where multiple disruptions occur, the reactive scheduling approach is invoked more than once.

Besides the reactive RCPSP, the reactive project scheduling without resource constraints has been studied several times (Klastorin and Mitchell, 2013; Zhu et al., 2007). Zhu et al. (2007) propose a two-stage stochastic programming approach for project planning under duration uncertainty. The first stage problem (SP) is the minimization of an expected cost that consists of two parts: the weighted sum of the completion times and the sum of the expected total weighted deviations from the targeted completion times over all scenarios. The former part, the weighted sum of the completion times, is included directly in the objective function whereas the latter one is computed indirectly by solving a second stage problem (SSP). In the SSP, for every combination of a scenario and a vector of targeted completion times a total weighted earliness and tardiness objective is computed to be minimized. Zhu et al. (2007) also include the cost of crashing the activities into the SSP in two ways (SSP1 and SSP2). In the SSP1, they add the cost of crashing as a constraint where the total crashing costs must be less than or equal to a predefined total crashing budget and in the SSP2, they add the cost of crashing to the objective function. Zhu et al. (2007) propose an LP-based heuristic for SSP1 and an efficient exact algorithm for SSP2. Klastorin and Mitchell (2013) propose a multi-stage *stochastic dynamic programming* (SDP) approach, tailored for the time/cost trade-off planning problem[13], to both find a stable baseline schedule and modify the baseline schedule when conflicts occur. They consider a multi-criteria objective function that covers labor costs, indirect costs and overhead costs.

[12]The matchup scheduling is a local rescheduling approach. The idea behind the matchup scheduling is to reschedule everything before a matchup point which is incrementally extended until a solution is constructed.

[13]For more details about the time/cost trade-off planning problem we refer to Demeulemeester and Herroelen (2002)

Chapter 2

The proactive and reactive resource-constrained project scheduling problem

Teachers can change lives with just the right mix of chalk and challenges.

- Joyce Meyer

Traditionally, a proactive and reactive project scheduling approach is a two stage approach. The first stage is to construct a schedule, which is called the *baseline* schedule, that is as robust as possible against a certain type of uncertainty (different types of uncertainty, such as activity duration uncertainty and resource availability uncertainty, have been introduced by several authors in the literature (Demeulemeester and Herroelen, 2011)). The second stage is to reschedule (react) reasonably whenever a *conflict* in the ongoing schedule occurs. A conflict refers to the situation where the schedule is no longer feasible.

A preliminary version of this chapter appeared as FEB Research Report KBI_1613 at KU Leuven (Davari and Demeulemeester, 2016a). This work has also been submitted for publication.

Despite the popularity of the traditional proactive and reactive procedures, we noticed that in almost all of the studies, the authors forgot two important aspects. First, these authors ignored the impact of the choice of the reactive scheduling policy on the optimality of the baseline schedule and vice versa. The only exception is the study by Deblaere et al. (2011c) where the proactive and reactive scheduling problems are solved simultaneously using a two-stage approach which outputs a baseline schedule and a corresponding reasonably efficient early-start policy. Not only does the procedure provide a simulation-based heuristic solution, but also the reactive policy is selected from a class of early start policies which extremely limits the flexibility of the procedure. Second, these authors simply assume that the number of reactions does not have any effect on the robustness of the baseline schedule and on that of the reactive policy. In fact, this assumption is inaccurate because of the same reasons that motivate the necessity of the baseline schedule. Moreover, we have to consider that not only each reaction costs some money, but it also damages the business credibility of the contractor.

What we suggest is an alternative method that resembles many robust optimization methods in the literature (Gabrel et al., 2014), specially *recoverable robust optimization* (Liebchen et al., 2009) and *adjustable robust optimization* (Ben-Tal et al., 2003; Shapiro, 2011). We try to tackle a *proactive and reactive resource-constrained project scheduling problem* (PR-RCPSP) considering the aforementioned forgotten aspects in project scheduling. The basic idea behind PR-RCPSP is to select a schedule from a given set of previously generated schedules and then foresee all required transitions to other schedules from the same set for each possible resource or precedence infeasibility. The combination of a baseline schedule and a set of required transitions is called a *proactive and reactive policy*. The detailed definition of the proactive-and-reactive policy is given in the next section.

In robust scheduling, the idea of generating multiple schedules is not new. For example, the *contingent scheduling approach*, which has been classified as a proactive scheduling approach (Chaari et al., 2014; Herroelen and Leus, 2005), is based on the generation of multiple baseline schedules. Another example is the very recent research by Akkan et al. (2016) in which a pool of solutions is generated and different neighborhoods of solutions are evaluated.

A schedule $\mathbf{s}$ is called infeasible if at least one activity i cannot be started at s_i without violating any resource or precedence constraint.

With respect to the current literature, the novelty of our contribution is three-fold. First, we introduce a novel way of reacting to conflicts. Second, we model proactive and reactive project scheduling as a single problem where the objective function includes the costs of reactions and is to find an optimal proactive-and-reactive policy. Third, we introduce four dynamic programming approaches (Models 1-4) that can solve the problem over different subsets of policies. Note that Model 1, which is outperformed by Models 2-4, is an intermediate model that simplifies the understanding of Models 2-4.

2.1 Definition and problem statement

In this section, we introduce the proactive and reactive resource-constrained project scheduling problem. The instances of the PR-RCPSP are very similar to the instances of the deterministic RCPSP with the only difference in the nature of the activity durations. In PR-RCPSP, each activity $i \in N \backslash \{0, n+1\}$ has an independent stochastic integer duration $\tilde{p}_i$ which follows a discrete and bounded distribution ($p_i^{\min} \leq \tilde{p}_i \leq p_i^{\max}$). Notice that $\tilde{p}_0 = \tilde{p}_{n+1} = 0$. We assume that the realized duration of the activity is only known when the execution of the activity is completed. The vector $\tilde{\mathbf{p}} = (\tilde{p}_0, \tilde{p}_1, ..., \tilde{p}_{n+1})$ can be represented by a finite supporting set $\mathfrak{P} = \{\mathbf{p}^1, ..., \mathbf{p}^{|\mathfrak{P}|}\}$ of realizations where each realization $\mathbf{p}^l$ is represented by a vector $\mathbf{p}^l = (p_0^l, p_1^l, ..., p_{(n+1)}^l) \in \mathfrak{P}$ of realized durations. The probability of occurrence of the realization $\mathbf{p}^l$ is shown by $\pi(\tilde{\mathbf{p}} = \mathbf{p}^l)$. Beware that $\mathfrak{P}$ can be a very large set, and as such, is only used in this section to facilitate introducing a compact formulation. Instead, our solution methodologies directly use the information provided by the given discrete distributions. Note that the methods proposed in this chapter are designed for the settings where activity durations are independent and must not be used in situations where activity durations are dependent.

2.1.1 Solution representation

A single schedule $\mathbf{s}$, which is a vector of starting times, cannot represent a solution because activity durations are stochastic. Instead, solutions are *proactive-and-reactive policies* (PR-policy). We introduce two different descriptions of a PR-policy in Section 2.1.1.1 and Section 2.1.1.2.

2.1.1.1 Rule-based description

A PR-policy Π is described by a set of decisions rules that dictate certain transitions among schedules. This set of decision rules that describe PR-policy Π is referred to as the *rule-based description* of PR-policy Π. At each decision moment, some information is become known, *i.e.*, the set of completed activities, the set of ongoing activities, the starting times of the ongoing and completed activities and the current execution time. This information defines the *state of the execution*. Each decision in PR-policy Π is associated with a certain state of execution. We assume that PR-policies behave consistently, *i.e.*, given the same pieces of information (state of execution), a PR-policy always makes the same decisions. In the following, we discuss the relation between a PR-policy and a traditional proactive and reactive scheduling solution.

In most reactive scheduling methods in the literature, a *reaction* is a set of rules that dictate a certain rescheduling of activities in order to resolve conflicts. Alternatively, we consider a *reaction* as the transition from one schedule to another schedule. Let $U(\mathbf{s}, t)$ represent the set of not yet started activities in schedule $\mathbf{s}$ at time t. A transition from schedule $\mathbf{s}$ to schedule $\mathbf{s}'$ is acceptable if and only if $U(\mathbf{s}, t) = U(\mathbf{s}', t)$ and $s_i = s'_i$ for all $i \in N \setminus U(\mathbf{s}, t)$.

A PR-policy Π not only determines reactions, but also selects one schedule as the baseline schedule. Let us consider the dummy schedule $\mathbf{s}^0 = (0, 0, \ldots, 0)$. This schedule, that is by definition infeasible to any given realization, represents the start of the project where no baseline schedule is present and no information is available. A PR-policy Π determines the baseline schedule, shown by $\mathbf{s}^{[0]\Pi}$, as the result of a reaction from $\mathbf{s}^0$ at time 0.

Let $\mathcal{S}(\mathbf{p}^l)$ be the set of all feasible schedules for a given realization $\mathbf{p}^l$. To address all possible schedules, we introduce the set $\mathcal{S}$ of schedules that can be constructed as follows $\mathcal{S} = \mathcal{S}(\mathbf{p}^1) \cup \cdots \cup \mathcal{S}(\mathbf{p}^{|\mathfrak{P}|})$. Since in the general case, $\mathcal{S}$ is a very large set, we define a much smaller set $\mathbf{S} \subset \mathcal{S}$ as the set of schedules. For practical reasons, we use the set $\mathbf{S}$ rather than $\mathcal{S}$ in our models (Models 1-4) in Section 2.2. Although the choice of the set $\mathbf{S}$ has a big influence on the performance of Models 1-4, it is not the focus of this chapter to find the best sets of schedules. However, in Section 2.2.1.5, we will explain a few heuristic approaches that construct reasonably good sets of schedules.

In this research, we merely consider PR-policies that dictate transitions among schedules only when conflicts occur. Such transitions are

referred to as *conflict-based* transitions and PR-policies that consist of only conflict-based transitions are called conflict-based PR-policies. In the remainder of this chapter, for the sake of simplicity and without loss of generality, we simply refer to conflict-based PR-policies as PR-policies unless expressly stated otherwise (for example see Section 2.4).

2.1.1.2 Chain-based description

In order to start the execution of the project, a PR-policy Π selects a baseline schedule ($\mathbf{s}^{[0]_\Pi}$). The execution is continued until the first conflict occurs. Since it is not known which realization will occur, PR-policy Π uses only the information that is available in the current state of the execution to transit to a second schedule. After that, the execution is continued until another conflict occurs. Then, PR-policy Π enforces another transition to a third schedule. PR-policy Π makes a series of transitions until either all conflicts are resolved or there is no schedule to which a transition from the current schedule is possible. The latter situation is referred to as a *deadend*. Notice that deadends only occur when we have a limited number of schedules in our set of schedules.

For each realization $\mathbf{p}^l$ (in case it occurs), PR-policy Π enforces a chain (series) of $\nu_{\Pi,l}$ reactions. This chain is denoted by $\Phi_{\Pi,l}$:

$$\Phi_{\Pi,l} : \mathbf{s}^{[0]_\Pi} \xrightarrow{t=t_1} \mathbf{s}^{[1]_{\Pi,l}} \xrightarrow{t=t_2} \mathbf{s}^{[2]_{\Pi,l}} \xrightarrow{t=t_3} \dots \xrightarrow{t=t_{\nu_{\Pi,l}}} \mathbf{s}^{[\nu_{\Pi,l}]_{\Pi,l}}.$$

The occurred realization and its associated chain of reactions are known only when the execution of the project has ended (either all activities are completed or a deadend is reached). Since PR-policies are consistent, the baseline schedule must be the same for all chains (realizations).

A chain with a deadend is called a *deadchain*. We introduce the parameter $\gamma_{\Pi,l}$ which equals one if $\Phi_{\Pi,l}$ is a deadchain and equals zero otherwise. We compute the incurred combined cost of PR-policy Π for each chain $\Phi_{\Pi,l}$. This combined cost, denoted by $f(\Pi, l)$, is also used as a measure for stability and robustness in Section 2.3:

$$f(\Pi,l) = w_b \times \mathbf{s}_{n+1}^{[0]_\Pi} + \sum_{k=1}^{\nu_{\Pi,l}} \left(\sum_{i \in U(\mathbf{s}^{[k-1]_{\Pi,l}}, t_k)} w_{ik} |\mathbf{s}_i^{[k]_{\Pi,l}} - \mathbf{s}_i^{[k-1]_{\Pi,l}}| + w_r \right) + \gamma_{\Pi,l} M.$$

where $w_b \geq 0$ is the cost per unit time for the completion time of the baseline schedule, $w_r \geq 0$ is the fixed cost incurred with each reaction, $w_{ik} \geq 0$ is the stability cost of activity i in the kth reaction and M represents the penalty of having a deadend. This cost function consists of three parts: the cost of the baseline schedule, the cost of a series of reactions and the cost of having a deadend.

A PR-policy Π, which is described by a set of decision rules, can also be described by its associated set of chains:

$$\Pi : \{\Phi_{\Pi,1}, \Phi_{\Pi,2}, ..., \Phi_{\Pi,|\mathfrak{P}|}\}.$$

This set of chains, which describes Π, is referred to as the *chain-based description* of Π. This description enables us to compute the expected combined cost of PR-policy Π:

$$\sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi, l).$$

Notice that the occurring realization is not known until the end of the execution and therefore the chain-based description, unlike its rule-based counterpart, is too confusing to be used in practice. However, it can be perfectly used in our problem formulation.

A PR-policy Π *includes* a schedule if that schedule has been a part of at least one of the chains that describe Π. It might be interesting to know what the worst case size of a PR-policy is, *i.e.*, the number of schedules that are included in that PR-policy. Let $\mathfrak{m}_i$ be the number of modes (*i.e.*, the number of different durations) of activity i.

Theorem 2.1. *The size of the chain-based description of a PR-policy is bounded by* $|\mathfrak{P}| \sum_{i \in N} \mathfrak{m}_i$.

Proof. Each PR-policy includes $|\mathfrak{P}|$ chains. This number, though, can be very large. For each chain (or realization), we can have at most $\sum_{i \in N} \mathfrak{m}_i$ reactions. Therefore, we infer that both the maximum number of schedules that are included and the maximum number of reactions that are needed in a PR-policy is bounded by $|\mathfrak{P}| \sum_{i \in N} \mathfrak{m}_i$. □

2.1.2 Conceptual formulation

We define $\mathbf{\Pi}$ as the set of all possible PR-policies (*i.e.*, the set of PR-policies that can be constructed by $\mathcal{S}$). Our problem (PR-RCPSP) which

is denoted for simplicity by P is formulated as follows:

$$\mathrm{P}: \min_{\Pi \in \mathbf{\Pi}} \sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi, l).$$

Theorem 2.2. P *is an NP-hard problem in the strong sense.*

Proof. We prove the NP-hardness of our problems by reduction from the deterministic RCPSP. Consider an instance of the deterministic RCPSP with activity duration vector $\mathbf{p}^{[det]}$. Set $w_b = 1$, $\pi(\tilde{p}_i = p) = 1$ if $p = p_i^{[det]}$ and $\pi(\tilde{p}_i = p) = 0$ if $p \neq p_i^{[det]}$. Since the durations are deterministic, no reaction will be in the optimal policy for this instance. Therefore, P finds a baseline schedule with the minimum makespan. □

P is a very difficult problem to solve, even for very small instances. We propose to solve P_1 which includes a smaller set of policies:

$$\mathrm{P}_1: \min_{\Pi \in \mathbf{\Pi}_1} \sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi, l).$$

In the above formulation, $\mathbf{\Pi}_1$ is the set of all PR-policies that only include the schedules in $\mathbf{S}$. Model 1 which is proposed in Section 2.2.1 optimally solves P_1 in a reasonable computation time where $|\mathbf{S}| \leq 2000$. Readers might question why we limit ourselves to PR-polices which select schedules only from $\mathbf{S}$. Selecting schedules only from $\mathbf{S}$ may lead to inflexible reactions and thus a high combined cost specially when $|\mathbf{S}|$ is too small. Therefore, we will introduce P_2, P_3 and P_4 in Section 2.2.2-2.2.4 that search over much larger classes of PR-policies and are optimally solved by Model 2, Model 3 and Model 4.

2.1.3 Example project

Let us describe an *example project* that provides a better understanding of our problem. Note that this example project will also be used and extended in other sections of this chapter and the next two chapters. Our example project includes 8 real activities and one resource type with an availability of 8. Figure 2.1 depicts the precedence relations and the resource requirements of activities of the project. Each node represents an activity, each arc represents a precedence relation and the number above each node shows the resource requirement for that activity.

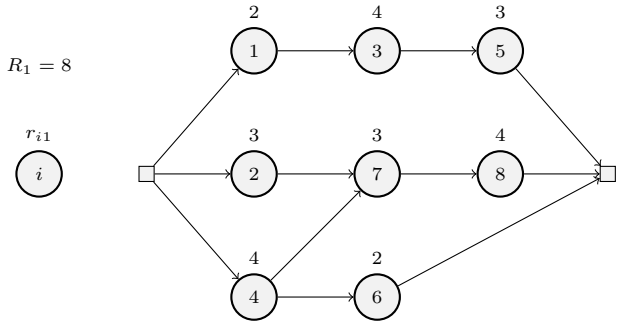


Figure 2.1: A copy of Figure 1.1.

	$\hat{p}_i$	$\pi(\tilde{p}_i = \hat{p}_i + \epsilon)$			$w_{i,0}$
		$\epsilon = -1$	$\epsilon = 0$	$\epsilon = +1$	
$\tilde{p}_0$	0	0	1	0	-
$\tilde{p}_1$	2	0.4	**0.4**	0.2	4
$\tilde{p}_2$	7	0.3	0.5	**0.2**	4
$\tilde{p}_3$	3	0	**0.6**	0.4	7
$\tilde{p}_4$	4	0.1	0.5	**0.4**	1
$\tilde{p}_5$	8	**0.2**	0.8	0	4
$\tilde{p}_6$	6	**0.4**	0.6	0	1
$\tilde{p}_7$	4	0.5	**0.5**	0	1
$\tilde{p}_8$	2	0	**0.7**	0.3	1
$\tilde{p}_9$	0	0	1	0	38

Table 2.1: The distribution of activity durations and the weights of the activities for the first reaction.

The activity durations for this example follow the distributions given in Table 2.1. In this table, $\hat{p}_i$ denotes the discrete average duration and ϵ is the possible deviation from this average value. Each fractional value represents the probability that $\tilde{p}_i = \hat{p}_i + \epsilon$. For instance, the probability that $\tilde{p}_1 = 1$ is 0.4. The probabilities in each row must obviously sum up to 1. Table 2.1 also include the stability cost of each activity. We assume that $w_{i,k} = w_{i,0}$ for all $k = 0, 1, ..., nm_{\max}$.

For this example, there are $2^5 \times 3^3 = 864$ realizations. Each realization occurs with a certain computable probability of occurrence. For example, $\mathbf{p}^1 = (0, 2, 8, 3, 5, 7, 5, 4, 2, 0)$ is a vector of durations which represents a

	$\mathbf{s}^k$									
	$\mathbf{s}^1$	$\mathbf{s}^2$	$\mathbf{s}^3$	$\mathbf{s}^4$	$\mathbf{s}^5$	$\mathbf{s}^6$	$\mathbf{s}^7$	$\mathbf{s}^8$	$\mathbf{s}^9$	$\mathbf{s}^{10}$
s_0^k	0	0	0	0	0	0	0	0	0	0
s_1^k	0	0	0	0	0	0	0	0	0	0
s_2^k	1	1	0	1	5	0	7	4	2	7
s_3^k	3	3	4	4	3	3	3	3	5	5
s_4^k	0	0	4	0	0	7	0	0	0	9
s_5^k	6	6	7	7	7	7	7	7	9	14
s_6^k	6	6	7	7	7	12	5	7	9	14
s_7^k	7	8	7	8	12	12	14	12	11	15
s_8^k	11	13	13	12	15	15	17	15	15	20
s_9^k	13	15	15	15	17	18	19	18	18	23

Table 2.2: A given set **S** for the example project.

single realization and its chance of occurrence is approximately 0.054% (the corresponding probabilities are indicated in bold in Table 2.1).

The set of schedules (**S**) that is given in Table 2.2 contains only 10 schedules. The scheduling parameters w_b, w_r and M are given as follows: $w_b = 40$, $w_r = 20$ and $M = 1000$.

Consider a PR-policy Π_1 with the following rule-based description:

- The baseline schedule is $\mathbf{s}^9$.
- At time 2 if the current schedule is $\mathbf{s}^9$ and activities 1 and 4 are ongoing, then react to schedule $\mathbf{s}^8$.
- At time 4 if the current schedule is $\mathbf{s}^8$ and activities 3 and 4 are ongoing, then react to schedule $\mathbf{s}^5$.

While the rule-based description is suitable for managerial purposes, a chain-based description is needed to compute the cost of the policy. The chain-based description of PR-policy Π_1 is given as follows:

$$\Pi_1 : \begin{cases} \Phi_{\Pi_1,l_1} : \mathbf{s}^9 & \mathbf{p}^{l_1} \in \mathfrak{P}_1 \\ \Phi_{\Pi_1,l_2} : \mathbf{s}^9 \xrightarrow{t=2} \mathbf{s}^8 & \mathbf{p}^{l_2} \in \mathfrak{P}_2 \\ \Phi_{\Pi_1,l_3} : \mathbf{s}^9 \xrightarrow{t=2} \mathbf{s}^8 & \mathbf{p}^{l_3} \in \mathfrak{P}_3 \\ \Phi_{\Pi_1,l_4} : \mathbf{s}^9 \xrightarrow{t=2} \mathbf{s}^8 \xrightarrow{t=4} \mathbf{s}^5 & \mathbf{p}^{l_4} \in \mathfrak{P}_4 \\ \Phi_{\Pi_1,l_5} : \mathbf{s}^9 \xrightarrow{t=2} \mathbf{s}^8 \xrightarrow{t=4} \mathbf{s}^5 & \mathbf{p}^{l_5} \in \mathfrak{P}_5 \end{cases}$$

where

$$\mathfrak{P}_1 = \{\mathbf{p}^l | p_1^l \leq 2, l = 1, ..., |\mathfrak{P}|\},$$
$$\mathfrak{P}_2 = \{\mathbf{p}^l | p_1^l > 2, p_4^l \leq 4, \mathbf{s}^8 \text{ is infeasible for } \mathbf{p}^l, l = 1, ..., |\mathfrak{P}|\},$$
$$\mathfrak{P}_3 = \{\mathbf{p}^l | p_1^l > 2, p_4^l \leq 4, \mathbf{s}^8 \text{ is feasible for } \mathbf{p}^l, l = 1, ..., |\mathfrak{P}|\},$$
$$\mathfrak{P}_4 = \{\mathbf{p}^l | p_1^l > 2, p_4^l > 4, \mathbf{s}^5 \text{ is infeasible for } \mathbf{p}^l, l = 1, ..., |\mathfrak{P}|\},$$
$$\mathfrak{P}_5 = \{\mathbf{p}^l | p_1^l > 2, p_4^l > 4, \mathbf{s}^5 \text{ is feasible for } \mathbf{p}^l, l = 1, ..., |\mathfrak{P}|\},$$

and $\mathfrak{P}_1 \cup \mathfrak{P}_2 \cup \mathfrak{P}_3 \cup \mathfrak{P}_4 \cup \mathfrak{P}_5 = \mathfrak{P}$. Note that the total number of chains is 864. Each chain Φ_{Π_1, l_x} in the above description represents a set of identical chains. For example, Φ_{Π_1, l_1} represents a set of 576 chains associated with all realizations $\mathbf{p}^l \in \mathfrak{P}_1$ ($|\mathfrak{P}_1| = 576$).

PR-policy Π_1 selects $\mathbf{s}^9$ as the baseline schedule. If $\mathbf{p}^1$ occurs, $\mathbf{s}^9$ never becomes infeasible and thus no reaction is needed. However, if $\mathbf{p}^2 = (0, 3, 8, 3, 4, 8, 5, 4, 2, 0)$ occurs, $\mathbf{s}^9$ becomes infeasible at time 2 (Figure 2.2(a)) and Π_1 dictates a transition from $\mathbf{s}^9$ to $\mathbf{s}^8$ to resolve the infeasibility. The associated Gantt charts are depicted in Figure 2.2. The cost of this reaction is computed as follows:

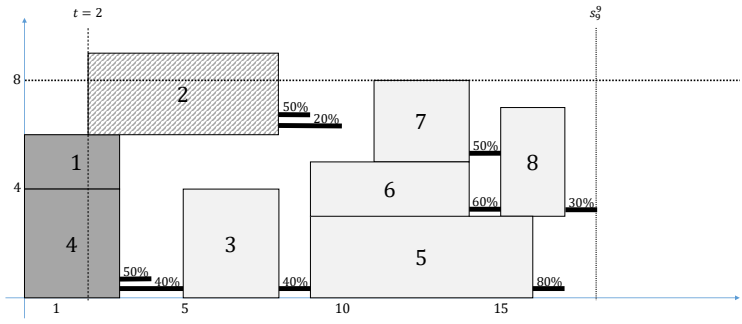
$$\sum_{i \in U(\mathbf{s}^9, 2)} w_{i,1} |s_i^8 - s_i^9| + w_r = |4 - 2| \times 4 + |3 - 5| \times 7 + |7 - 9| \times 4 + |7 - 9| \times 1 + |12 - 11| \times 1 + |15 - 15| \times 1 + |18 - 18| \times 38 + 20 = 53.$$

If $\mathbf{p}^3 = (0, 3, 8, 3, 5, 8, 5, 4, 2, 0)$ occurs, Π_1 dictates a reaction from $\mathbf{s}^9$ to $\mathbf{s}^8$ to resolve the infeasibility at time 2 and a reaction from $\mathbf{s}^8$ to $\mathbf{s}^5$ to resolve the infeasibility at time 4 (the associated Gantt charts are depicted in Figure 2.2). The cost of the second reaction is computed as follows:

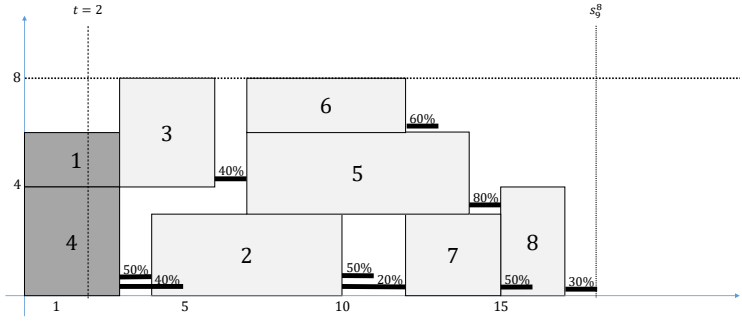
$$\sum_{i \in U(\mathbf{s}^8, 4)} w_{i,2} |s_i^5 - s_i^8| + w_r = 62.$$

For each realization $\mathbf{p}^l$ in $\mathfrak{P}_1$, we compute the cost of the associated chain as follows: $f(\Pi_1, l) = 40 \times 18 = 720$. The cumulative probability of occurrence of realizations in $\mathfrak{P}_1$ equals $\sum_{l \in \mathfrak{P}_1} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) = 0.80$. Similarly we compute:

- for each $\mathbf{p}^l \in \mathfrak{P}_2$, $f(\Pi_1, l) = 720 + 53 + 1000 = 1773$,
- for each $\mathbf{p}^l \in \mathfrak{P}_3$, $f(\Pi_1, l) = 720 + 53 = 773$,

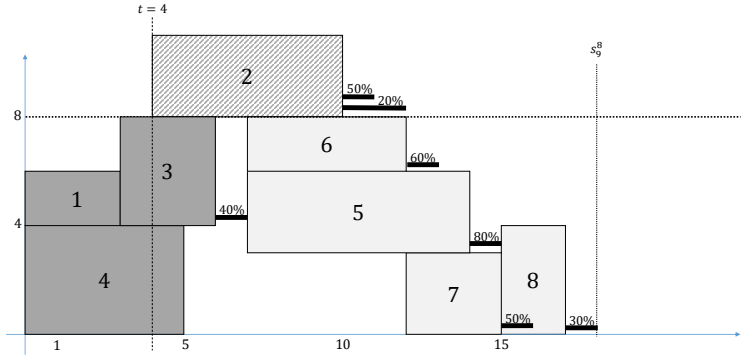


(a) Gantt chart for schedule $\mathbf{s}^9$ which becomes infeasible at time 2 for realizations $\mathbf{p}^2$ and $\mathbf{p}^3$

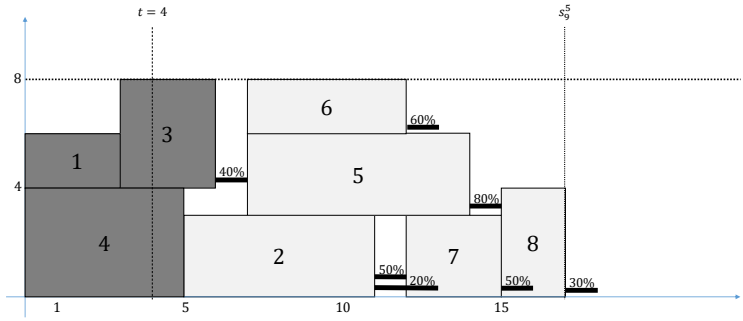


(b) Gantt chart for schedule $\mathbf{s}^8$ which is feasible at time 2 for realizations $\mathbf{p}^2$ and $\mathbf{p}^3$

Figure 2.2: An example reaction. The infeasibility is resolved by a transition from schedule $\mathbf{s}^9$ to schedule $\mathbf{s}^8$ at time 2. The length of each box indicates the minimum possible duration for the associated activity. Other possible durations of an activity are indicated by horizontal bars together with their probabilities of occurrence. Note that at time t, the lengths of the associated boxes of finished activities represent their realized duration. Also notice that the minimum length of the box associated with an ongoing activity cannot be less than or equal to its realized duration so far.



(c) Gantt chart for schedule $\mathbf{s}^8$ which becomes infeasible at time 4 for realization $\mathbf{p}^3$



(d) Gantt chart for schedule $\mathbf{s}^5$ which is feasible at time 4 for realization $\mathbf{p}^3$

Figure 2.2: An example reaction (continued).

- for each $\mathbf{p}^l \in \mathfrak{P}_4$, $f(\Pi_1, l) = 720 + 53 + 62 + 1000 = 1835$ and
- for each $\mathbf{p}^l \in \mathfrak{P}_5$, $f(\Pi_1, l) = 720 + 53 + 62 = 835$.

Also, $\sum_{l\in\mathfrak{P}_2} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) = 0.06$, $\sum_{l\in\mathfrak{P}_3} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) = 0.06$, $\sum_{l\in\mathfrak{P}_4} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) = 0.0576$ and $\sum_{l\in\mathfrak{P}_5} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) = 0.0224$. The expected combined cost of PR-policy Π_1 is

$$\sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi_1, l) = 0.80 \times 720 + 0.06 \times (1773 + 773) + 0.0576 \times 1835 + 0.0224 \times 835 = 853.16.$$

PR-policy Π_1 happens to be the optimal PR-policy for P_1.

2.2 Solution methodology

Problem P can be modeled as a *Markov decision process* (MDP). We propose four MDPs in this section, namely Model 1, Model 2, Model 3 and Model 4. Model 1, which will be explained in Section 2.2.1, optimally solves P_1 for a given set of schedules. Nevertheless, this model only searches over a limited set of PR-policies. Therefore, we also provide Models 2-4 in the following subsections, which search over much larger sets of PR-policies than that of P_1.

This is not the first time MDPs are used to deal with stochasticity in project scheduling. We refer interested readers to Creemers et al. (2010) and Creemers (2015).

2.2.1 Model 1

As we mentioned above, Model 1 is an MDP that optimally solves P_1. Our model description is structured as follows. First, the state representation is given. Then, the transitions among states are introduced. Next, the recursion system is described. After that, we introduce a graph representation by means of an example and finally, an algorithm is proposed to generate the set of schedules.

2.2.1.1 State representation

The combination $(\mathbf{s}, t, O, \nu)$ represents a state in Model 1 where $\mathbf{s}$ represents the current schedule, t is the current time, O denotes the set of ongoing activities at time t and ν is the total number of reactions that previously occurred upon entrance of the state.

States are labeled either as feasible or infeasible. Let $J(\mathbf{s}, t)$ be the set of all activities in $\mathbf{s}$ that are supposed to be started at time t.

Definition 2.1 (Feasible vs infeasible states)**.** *A state $(\mathbf{s}, t, O, \nu)$ is feasible if it is possible to execute all activities in $J(\mathbf{s}, t) \cup O$ in parallel and is infeasible otherwise.*

Example. *Figure 2.2(a) depicts the Gantt chart of $\mathbf{s}^9$ at time 2 for realization $\mathbf{p}^2$. The associated state is $(\mathbf{s}^9, 2, \{1, 4\}, 0)$. This state is an*

infeasible state since the set of activities $J(\mathbf{s}^9, 2) \cup \{1, 4\} = \{1, 2, 4\}$ *cannot be executed in parallel. Figure 2.2(b) depicts the Gantt chart of* $\mathbf{s}^8$ *at time* 2 *for realization* $\mathbf{p}^2$*. The associated state is* $(\mathbf{s}^8, 2, \{1, 4\}, 1)$ *which is a feasible state because the set of activities* $J(\mathbf{s}^8, 2) \cup \{1, 4\} = \{1, 4\}$ *can be executed in parallel.*

2.2.1.2 Transitions

Upon leaving a state, we enter another state. Entering state $(\mathbf{s}, t, O, \nu)$ means that schedule $\mathbf{s}$ is considered for execution, the current time is t, the activities in O are ongoing and we have already reacted to conflicts ν times. Depending on whether we face a conflict (the current state is infeasible) or not (the current state is feasible), and also depending on the realization and the PR-policy we choose, we could enter a different state.

A transition between two states is possible if and only if an arc exists between those two states. The states are connected via two types of arcs: *chance* arcs and *decision* arcs. Chance arcs leave feasible states whereas decision arcs leave infeasible states.

If state $(\mathbf{s}, t, O, \nu)$ is feasible, it means no conflict is happening and there is no need for any reaction at that decision moment. In this case, a transition is required to a new state $(\mathbf{s}, t', O', \nu)$ where t' is the next decision moment in $\mathbf{s}$ after t and $O' \subseteq O \cup J(\mathbf{s}, t)$. The set O' might differ depending on which realization occurs. Therefore, the number of chance arcs leaving a state might be more than one. Upon leaving a feasible state, we may enter one of the states to which a chance arc exists from the left state. A chance arc that connects $(\mathbf{s}, t, O, \nu)$ to $(\mathbf{s}, t', O', \nu)$ is denoted by $(\mathbf{s}, t \to t', O \to O', \nu)$. Because the probability of occurrence of each realization is known, we can compute the probability of transition over a certain chance arc $(\pi(\mathbf{s}, t \to t', O \to O', \nu))$ as follows:

$$\pi(\mathbf{s},t \to t', O \to O', \nu) = \prod_{i \in O \cap O'} \frac{\pi(s_i + \tilde{p}_i > t')}{\pi(s_i + \tilde{p}_i > t)} \times \prod_{i \in J(\mathbf{s},t) \cap O'} \pi(t + \tilde{p}_i > t')$$
$$\times \prod_{i \in O \setminus O'} \left(1 - \frac{\pi(s_i + \tilde{p}_i > t')}{\pi(s_i + \tilde{p}_i > t)}\right) \times \prod_{i \in J(\mathbf{s},t) \setminus O'} (1 - \pi(t + \tilde{p}_i > t')).$$

Note that the summation of the probabilities of chance arcs leaving a feasible state must equal 1. If $O = \emptyset$ and $J(\mathbf{s}, t) = \{n + 1\}$, then the execution has already completed and no chance arc leaves $(\mathbf{s}, t, O, \nu)$. In this case, $(\mathbf{s}, t, O, \nu)$ is an *endstate*.

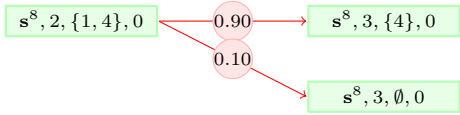


Figure 2.3: Chance arcs leaving $(\mathbf{s}^8, 2, \{1,4\}, 0)$.

Example. *Figure 2.2(b) depicts the Gantt chart associated with state* $(\mathbf{s}^8, 2, \{1,4\}, 1)$. *If we move to the next decision moment, which is time 3, depending on which realization occurs, we arrive to one of two different states:* $(\mathbf{s}^8, 3, \{4\}, 1)$ *or* $(\mathbf{s}^8, 3, \emptyset, 1)$. *Figure 2.3 demonstrates the chance arcs leaving state* $(\mathbf{s}^8, 3, \{1,4\}, 1)$. *We compute the probability of each chance arc leaving* $(\mathbf{s}^8, 3, \{1,4\}, 1)$ *as follows:*

$$\pi(\mathbf{s}^8, 2 \to 3, \{1,4\} \to \{4\}, 1) = \frac{0.90}{1} \times \left(1 - \frac{0}{0.2}\right) = 0.90,$$
$$\pi(\mathbf{s}^8, 2 \to 3, \{1,4\} \to \emptyset, 1) = \left(1 - \frac{0}{0.2}\right) \times \left(1 - \frac{0.90}{1}\right) = 0.10.$$

Upon leaving an infeasible state, we decide to transit to a feasible state. However, not all such transitions are valid. There does exist a decision arc from an infeasible state $(\mathbf{s}, t, O, \nu)$ to a feasible state $(\mathbf{s}', t, O, \nu+1)$ if and only if $U(\mathbf{s}, t) = U(\mathbf{s}', t)$ and $s_i = s'_i$ for all $i \in N \setminus U(\mathbf{s}, t)$. Figure 2.4 depicts a Gantt chart illustration of a valid transition from $(\mathbf{s}, t, O, \nu)$ to $(\mathbf{s}', t, O, \nu+1)$. In this Gantt chart representation of the schedules, F represents the finished activities, O represents the ongoing activities at time t and $U(\mathbf{s}, t)$ represents the activities in the schedule $\mathbf{s}$ that have not started their execution before time t. Note that although it might not be clear in the Gantt charts, the starting times of activities in F and O should be exactly the same for the two schedules. In reality, a transition from $\mathbf{s}$ to $\mathbf{s}'$ at time t means that the management team decides to continue the execution of the project according to schedule $\mathbf{s}'$ (instead of the infeasible schedule $\mathbf{s}$) from time t until a further conflict occurs. These transitions are called *normal transitions*. In the following subsections, we propose other types (namely, flexible, cut and wise transitions) of transitions.

For each infeasible state $(\mathbf{s}, t, O, \nu)$, we introduce $\Gamma_1(\mathbf{s}, t, O)$ that represents the set of all schedules to which a valid transition from $(\mathbf{s}, t, O, \nu)$ exists. In other words, if $S' \in \Gamma_1(\mathbf{s}, t, O)$, then there exists a transition from $(\mathbf{s}, t, O, \nu)$ to $(\mathbf{s}', t, O, \nu+1)$. The larger the set $\Gamma_1(\mathbf{s}, t, O)$, the more flexibility we have in reacting. We noticed that if $\mathbf{S}$ is a finite set, the set

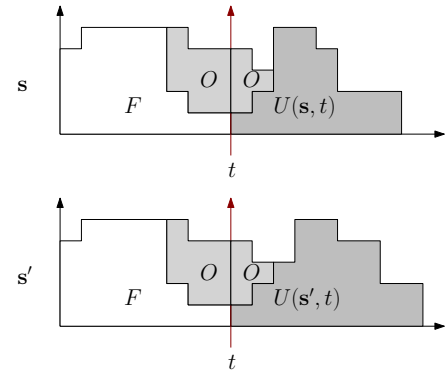


Figure 2.4: Transition from $\mathbf{s}$ to $\mathbf{s}'$ at time t.

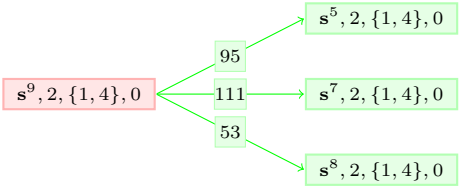


Figure 2.5: Decision arcs leaving $(\mathbf{s}^9, 2, \{1,4\}, 0)$.

$\Gamma_1(\mathbf{s}, t, O)$ might be an empty set for some combinations of $\mathbf{s}$, t and O. If a state $(\mathbf{s}, t, O, \nu)$ is infeasible and $\Gamma_1(\mathbf{s}, t, O) = \emptyset$, then we have no reaction possibility. Such a state is called a *deadstate*. Notice that a deadstate is the last state in a deadchain and its corresponding schedule is a deadend for the associated state of execution.

Example. *Consider the infeasible state* $(\mathbf{s}^9, 2, \{1,4\}, 0)$. *We compute the set of schedules that are valid candidates for reaction:* $\Gamma_1(\mathbf{s}^9, 2, \{1,4\}) = \{\mathbf{s}^5, \mathbf{s}^7, \mathbf{s}^8\}$. *Figure 2.5 demonstrates the decision arcs leaving the infeasible state* $(\mathbf{s}^9, 2, \{1,4\}, 0)$. *The number depicted on each arc represents the cost of the associated reaction. For example, the cost of reacting to schedule* $\mathbf{s}^5$ *is* 95.

2.2.1.3 Dynamic programming recursion

We introduce $d_1(\mathbf{s} \to \mathbf{s}', t, O, \nu \to \nu + 1)$ as the expected cost until the end of execution if we decide to transit from $(\mathbf{s}, t, O, \nu)$ to $(\mathbf{s}', t, O, \nu + 1)$

in Model 1. We also introduce $c_1(\mathbf{s}, t, O, \nu)$ as the expected cost until the end of execution upon leaving feasible state $(\mathbf{s}, t, O, \nu)$. First, we compute the expected cost until the end of the execution for each decision arc:

$$d_1(\mathbf{s} \to \mathbf{s}', t, O, \nu \to \nu+1) = w_r + \sum_{i \in U(\mathbf{s},t)} \left(w_{i(\nu+1)} |s_i' - s_i|\right) + c_1(\mathbf{s}', t, O, \nu+1). \tag{2.1}$$

Then, the best decision and its associated expected cost are computed as follows:

$$\mathbf{s}^* \in \underset{\mathbf{s}' \in \Gamma_1(\mathbf{s},t,O)}{\arg\min} \{d_1(\mathbf{s} \to \mathbf{s}', t, O, \nu \to \nu+1)\} \tag{2.2}$$

$$d_1^*(\mathbf{s}, t, O, \nu) = d_1(\mathbf{s} \to \mathbf{s}^*, t, O, \nu \to \nu+1). \tag{2.3}$$

Let $\mathcal{F}_1$ be the set of all feasible states and $\mathcal{I}_1$ be the set of all infeasible states in Model 1. Likewise, the set of all endstates and the set of all deadstates are denoted by $\mathcal{E}_1$ and $\mathcal{D}_1$, respectively. The function $c_1(\mathbf{s}, t, O, \nu)$ is computed as follows (remember that the expected cost of an endstate is zero):

$$\begin{aligned} c_1(\mathbf{s},t,O,\nu) = & \sum\nolimits_{(\mathbf{s},t',O',\nu) \in \mathcal{F}_1 \setminus \mathcal{E}_1} \pi(\mathbf{s}, t \to t', O \to O', \nu) * c_1(\mathbf{s}, t', O', \nu) \\ & + \sum\nolimits_{(\mathbf{s},t',O',\nu) \in \mathcal{I}_1 \setminus \mathcal{D}_1} \pi(\mathbf{s}, t \to t', O \to O', \nu) * d_1^*(\mathbf{s}, t', O', \nu) \\ & + \sum\nolimits_{(\mathbf{s},t',O',\nu) \in \mathcal{D}_1} \pi(\mathbf{s}, t \to t', O \to O', \nu) * M. \end{aligned} \tag{2.4}$$

Note that in the above equation, t' is always the next decision moment after t and O' is whatever set that satisfies $O' \subseteq O \cup J(\mathbf{s}, t)$.

We need to select one of the schedules in $\mathbf{S}$ as the baseline schedule. To that end, we can use the information provided while solving our model. The cost of choosing schedule $\mathbf{s}$ as the baseline schedule equals $c_1(\mathbf{s}, 0, \emptyset, 0) + w_b s_{n+1}$. Therefore, we select the baseline schedule ($\mathbf{s}^{\text{base}}$) as follows:

$$\mathbf{s}^{\text{base}} \in \underset{\mathbf{s} \in \mathbf{S}}{\arg\min} \{c_1(\mathbf{s}, 0, \emptyset, 0) + w_b s_{n+1}\}. \tag{2.5}$$

It is straightforward to see that the recursion system (2.1)-(2.5) optimally solves P_1 and the optimal objective value equals $c_1(\mathbf{s}^{\text{base}}, 0, \emptyset, 0) + w_b s_{n+1}^{\text{base}}$. Let $O_{\max}$ be the set of ongoing activities with maximum cardinality. The total number of states is in the order of $\text{O}(n 2^{|O_{\max}|} |\mathbf{S}| \sum_{i \in N} \mathfrak{m}_i)$

and the maximum number of arcs (chance arcs or decision arcs) leaving a state equals $\max\{2^{|O_{\max}|}, |\mathbf{S}|\}$. Therefore, Model 1 can be solved in $O(n^2 2^{|O_{\max}|}|\mathbf{S}| \sum_{i \in N} \mathfrak{m}_i \max\{2^{|O_{\max}|}, |\mathbf{S}|\})$ time.

2.2.1.4 Graph representation

In this section, we present a graph representation for Model 1 (Figure 2.6). Each state is represented by a node: green nodes represent feasible states, red nodes represent infeasible states and gray nodes represent deadstates. The number shown above each state $(\mathbf{s}, t, O, \nu)$ indicates the expected cost until the end of execution upon leaving, which equals $c_1(\mathbf{s}, t, O, \nu)$ for feasible states, $d_1^*(\mathbf{s}, t, O, \nu)$ for infeasible states or M for deadstates. Each transition (reaction) is represented by an arc: green arcs are decision arcs and red arcs are chance arcs. The number shown over each chance arc is the probability of crossing that chance arc. The number shown over each decision arc is either the cost of the baseline schedule (which is the case for the arcs leaving the start node) or the cost of the associated reaction (which is the case for the arcs leaving any infeasible state expect the start node). Note that the start node represents the situation where even the baseline schedule is not yet decided.

For the example project, the start node is incident to ten different feasible states, each representing one potential baseline schedule. Due to lack of space we only depict a small part of the network. For example, upon entering $(\mathbf{s}^1, 0, \emptyset, 0)$, activities 1 and 4 are started. In $\mathbf{s}^1$, the next decision moment after 0 is 1. Because activity 4 certainly takes longer than 1 time unit and activity 1 can last 1, 2 or 3 time units, either of the following two transitions can occur upon leaving $(\mathbf{s}^1, 0, \emptyset, 0)$: a transition to the infeasible state $(\mathbf{s}^1, 1, \{1, 4\}, 0)$, where activities 1 and 4 are ongoing at decision moment 1 and activity 2 is about to start, or a transition to the feasible state $(\mathbf{s}^1, 1, \{4\}, 0)$, where activity 4 is ongoing at decision moment 1 and activity 2 is about to start. The former transition occurs with a probability of 60% while the latter one occurs with a probability of 40%. Knowing the expected costs until the end of execution upon leaving the states $(\mathbf{s}^1, 1, \{1, 4\}, 0)$ and $(\mathbf{s}^1, 1, \{4\}, 0)$, we can simply compute the expected cost until the end of execution upon leaving the state $(\mathbf{s}^1, 0, \emptyset, 0)$ as follows:

$$\begin{aligned} c_1(\mathbf{s}^1, 0, \emptyset, 0) &= 0.40 \times c_1(\mathbf{s}^1, 1, \{4\}, 0) + 0.60 \times d_1^*(\mathbf{s}^1, 1, \{1, 4\}, 0) \\ &= 0.40 \times 985.15 + 0.60 \times 472.93 = 677.82. \end{aligned}$$

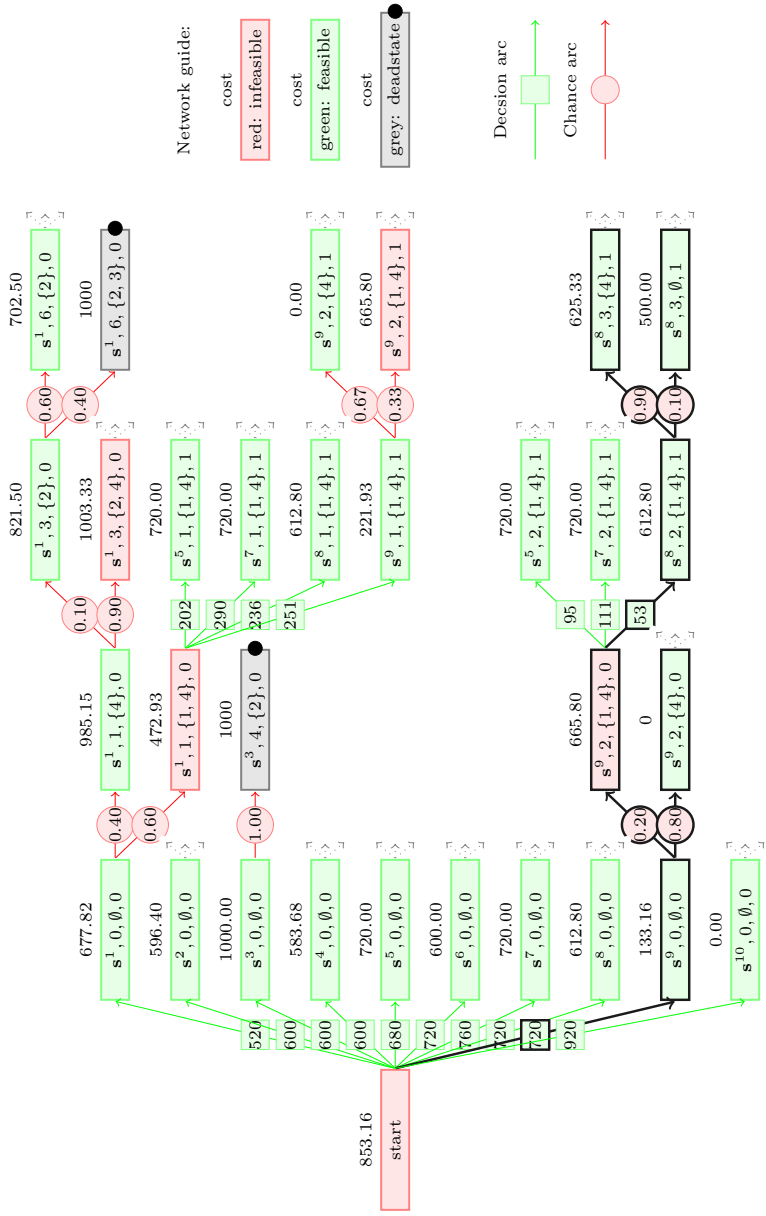


Figure 2.6: A part of the network of Model 1 associated with the example in Section 2.2.1.4.

Figure 2.6 also indicates in dark black part of an optimal PR-policy Π_1. PR-policy Π_1, as stated in the definition of PR-policy, enforces certain decisions for each infeasible state. For example, among all arcs leaving the start node, Π_1 chooses $(\mathbf{s}^9, 0, \emptyset, 0)$ and therefore $\mathbf{s}^{[0]_{\Pi_1}} = \mathbf{s}^9$. Also upon leaving $(\mathbf{s}^9, 2, \{1, 4\}, 0)$, which is an infeasible state and represents a conflict in schedule $\mathbf{s}^9$ at time 2 when activities 1 and 4 are ongoing and activity 2 is about to start, Π_1 dictates a transition to $(\mathbf{s}^8, 2, \{1, 4\}, 1)$. Notice that the expected cost until the end of execution upon leaving the state $(\mathbf{s}^9, 2, \{1, 4\}, 0)$ is computed as follows:

$$d_1^*(\mathbf{s}^9, 2, \{1, 4\}, 0) = \min\{612.80 + 53, 720 + 111, 720 + 95\} = 665.80.$$

2.2.1.5 The set of schedules

In Algorithm 2.1, we present a *pool generation procedure* (PGP) that outputs a set of schedules given a set of initial schedules. In this procedure, κ_1 is the exact number of schedules that will be generated. Due to the high computational complexity of our methods, we limit ourselves to $\kappa_1 \leq 2000$. A number of subprocedures are also used in PGP which will be explained in the following lines: *rndSchedule*($\mathbf{S}$) returns one schedule which is selected randomly from the set $\mathbf{S}$ of all already generated schedules, *rndRealization*($\mathfrak{P}$) returns a random realization, *infeas*($\mathbf{s}, \mathbf{p}^l, t$) returns *true* if schedule $\mathbf{s}$ becomes infeasible at decision moment t for realization $\mathbf{p}^l$ and returns *false* otherwise and *nextDM*($\mathbf{s}, t$) returns the next decision moment after t in schedule $\mathbf{s}$. The subprocedure *react*($\mathbf{s}, \mathbf{p}^l, t$) which reacts to the conflict at decision moment t is a *robust parallel schedule generation scheme* (RP-SGS) introduced by Van de Vonder et al. (2007a) where the duration vector is $\hat{\mathbf{p}}$ and the scheme's priority rule is the *earliest baseline activity starting time* (EBST).

Depending on what initial set we select, what value we choose for κ_1 and what is the reaction policy, PGP results in different sets of schedules. The initial set could be one of the following: 1) a set including one single schedule that is the optimal schedule for the deterministic case where the duration vector is $\hat{\mathbf{p}}$ (we refer to this set as DET), 2) a set including 13 schedules that are the outcome of the *starting time criticality* (STC) procedure proposed by Van de Vonder et al. (2008), each with a different α selected from $\{1, 1.025, 1.050, ..., 1.25, 1.275, 1.3\}$ (we refer to this set as STC) and 3) a set of fifty schedules generated by Algorithm 2.2 (we set $\kappa_2 = 50$ and $\kappa_3 = 4$). Notice that the notation α is borrowed from the notation system in Van de Vonder et al. (2008) and as such

Algorithm 2.1 Pool generation procedure (PGP)

Input: A set $\mathbf{S}^{\text{init}}$ of initial schedules

$\mathbf{S} \leftarrow \mathbf{S}^{\text{init}}$
$n_{\mathbf{S}} \leftarrow |\mathbf{S}^{\text{init}}|$
while $n_{\mathbf{S}} < \kappa_1$ **do**
  $\mathbf{s} \leftarrow \mathit{rndSchedule}(\mathbf{S})$
  $\mathbf{p}^l \leftarrow \mathit{rndRealization}(\mathfrak{P})$
  $t \leftarrow 0$
  while $t \leq s_{n+1}$ **do**
    if $\mathit{infeas}(\mathbf{s}, \mathbf{p}^l, t)$ **then**
      $\mathbf{s}' \leftarrow \mathit{react}(\mathbf{s}, \mathbf{p}^l, t)$
      $\mathbf{s} \leftarrow \mathbf{s}'$
      if $\mathbf{s} \notin \mathbf{S}$ **then**
        $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathbf{s}\}$
        $n_{\mathbf{S}} \leftarrow n_{\mathbf{S}} + 1$
      if $n_{\mathbf{S}} = \kappa_1$ **then**
        break
    $t \leftarrow \mathit{nextDM}(\mathbf{s}, t)$

Output: $\mathbf{S}$

Algorithm 2.2 An initial pool generation scheme

$\mathbf{S}^{\text{init}} \leftarrow \emptyset$
$n_{\mathbf{S}} \leftarrow 0$
while $n_{\mathbf{S}} < \kappa_2$ **do**
  generate κ_3 random realization $\mathbf{p}^1, ..., \mathbf{p}^{\kappa_3}$
  $\mathbf{p}^{\max} \leftarrow \max\{\mathbf{p}^1, ..., \mathbf{p}^{\kappa_3}\}$
  $\mathbf{s} \leftarrow DH(\mathbf{p}^{\max})$
  $\mathbf{S}^{\text{init}} \leftarrow \mathbf{S}^{\text{init}} \cup \{\mathbf{s}\}$
  $n_{\mathbf{S}} \leftarrow n_{\mathbf{S}} + 1$

Output: $\mathbf{S}^{\text{init}}$

must not be confused with $(1 - \alpha)$ that denotes the confidence level in Chapter 4. In Algorithm 2.2, the subprocedure $DH(\cdot)$ optimally solves the deterministic RCPSP using the branch-and-bound method proposed by Demeulemeester and Herroelen (1992, 1997) (we refer to this set as SMP which stands for sampling).

2.2.2 Model 2

In Section 2.2.1, Model 1 which optimally solves P_1 was proposed. The optimal PR-policy to P_1 has a reasonable combined cost only if the given set of schedules is very large. However, as we have already mentioned, Model 1 is not computationally tractable when the given set of schedules is large and thus we only can solve instances of P_1 where the given set of schedules is very small. This results in a high probability of having many deadends in the optimal PR-policy and leads to a very high optimal combined cost. This motivated us to investigate what would be an alternative model that searches over larger sets of PR-policies and yet be computationally tractable.

The poor performance of Model 1 is mainly because of its inflexibility in reactions. In Model 2, we try to relax some criteria to allow more flexibility in reactions. We noticed that it is only needed to know the current state of execution of the project rather than the starting times of the activities that have already been started. More specifically, in the new model, a transition from $(\mathbf{s}, t, O, \nu)$ to $(\mathbf{s}', t + de, O, \nu + 1)$ is possible if three conditions are met: 1) $s_i = s'_i + de, \forall i \in O$, 2) the equality $U(\mathbf{s}, t) = U(\mathbf{s}', t + de)$ holds and 3) either of the two criteria below is true:

- $t + de$ is a decision moment in $\mathbf{s}'$ and $(\mathbf{s}', t + de, O, \nu + 1)$ is feasible.
- $t + de$ is not a decision moment in $\mathbf{s}'$.

Note that de is whatever constant value (including 0). This transition is called a *flexible transition*.

Figure 2.7 illustrates the condition in which a flexible transition from $\mathbf{s}$ to $\mathbf{s}'$ is valid. In reality, a flexible transition results in a new schedule that might not exist within the set of schedules, but can be implied by the schedules in the set. In other words, the flexible transition from $(\mathbf{s}, t, O, \nu)$ to $(\mathbf{s}', t + de, O, \nu + 1)$ means that the managerial team decides to continue the execution of the project according to $\mathbf{s}^{\#}$ (the schedule at the very bottom of Figure 2.7), from time t until another conflict occurs in the schedule. The schedule $\mathbf{s}^{\#}$ might not exist in the set of all schedules, but it can be deduced from $\mathbf{s}$ and $\mathbf{s}'$ as explained below:

$$s_i^{\#} = \begin{cases} s_i & \text{if } s_i < t \\ s'_i - de & \text{if } s_i \geq t \end{cases}.$$

We introduce $\Gamma_2(\mathbf{s}, t, O)$ as the set of all pairs $(\mathbf{s}', de)$ for which a transition from $(\mathbf{s}, t, O, \nu)$ to $(\mathbf{s}', t + de, O, \nu + 1)$ is possible. The recursion

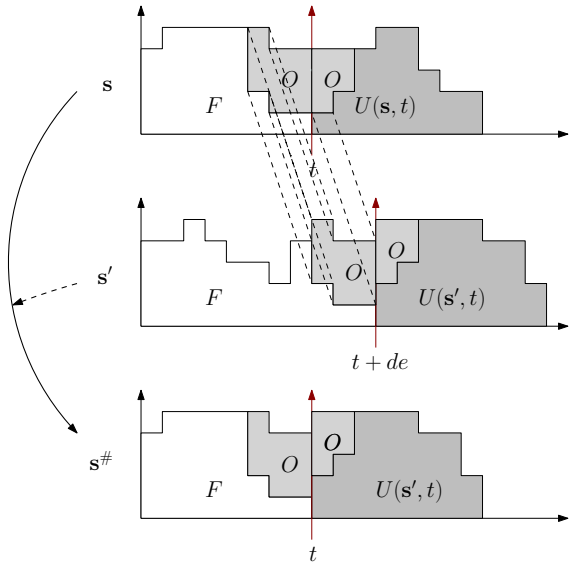


Figure 2.7: A flexible transition from $\mathbf{s}$ to $\mathbf{s}'$ at time t.

system for Model 2 can be computed in the same fashion as that of Model 1 except that, in Model 2, we use $\Gamma_2(\mathbf{s}, t, O)$ instead of $\Gamma_1(\mathbf{s}, t, O)$. We introduce $d_2(\mathbf{s} \to \mathbf{s}', t \to t + de, O, \nu \to \nu + 1)$ as the expected cost until the end of execution if we decide to transit from $(\mathbf{s}, t, O, \nu)$ to $(\mathbf{s}', t + de, O, \nu + 1)$ in Model 2 and $c_2(\mathbf{s}, t, O, \nu)$ as the corresponding expected cost until the end of execution upon leaving each feasible state $(\mathbf{s}, t, O, \nu)$.

$$d_2(\mathbf{s} \to \mathbf{s}', t \to t + de, O, \nu \to \nu + 1) = w_r + \sum_{i \in U(\mathbf{s},t)} \left(w_{i(\nu+1)} |s'_i + de - s_i|\right) + c_2(\mathbf{s}', t + de, O, \nu + 1) \quad (2.6)$$

Then, the cost associated with the best decision is computed as follows:

$$(\mathbf{s}^*, de^*) \in \underset{(\mathbf{s}', de) \in \Gamma_2(\mathbf{s}, t, O)}{\arg\min} \{d_2(\mathbf{s} \to \mathbf{s}', t \to t + de, O, \nu \to \nu + 1)\} \quad (2.7)$$

$$d_2^*(\mathbf{s}, t, O, \nu) = d_2(\mathbf{s} \to \mathbf{s}^*, t \to t + de^*, O, \nu \to \nu + 1). \quad (2.8)$$

We introduce the set $\mathcal{F}_2$ and $\mathcal{I}_2$ as the set of all feasible states and the set of all infeasible states for Model 2, respectively. Also, the set of

all endstates and the set of all deadstates are represented by $\mathcal{E}_2$ and $\mathcal{D}_2$, respectively. The function $c_2(\mathbf{s}, t, O, \nu)$ is computed as follows:

$$\begin{aligned} c_2(\mathbf{s},t,O,\nu) = &\sum\nolimits_{(\mathbf{s},t',O',\nu)\in\mathcal{F}_2\setminus\mathcal{E}_2} \pi(\mathbf{s}, t\to t', O\to O', \nu) * c_2(\mathbf{s}, t', O', \nu) \\ &+ \sum\nolimits_{(\mathbf{s},t',O',\nu)\in\mathcal{I}_2\setminus\mathcal{D}_2} \pi(\mathbf{s}, t\to t', O\to O', \nu) * d_2^*(\mathbf{s}, t', O', \nu) \\ &+ \sum\nolimits_{(\mathbf{s},t',O',\nu)\in\mathcal{D}_2} \pi(\mathbf{s}, t\to t', O\to O', \nu) * M. \end{aligned} \tag{2.9}$$

We select the baseline schedule ($\mathbf{s}^{\text{base}}$) as follows:

$$\mathbf{s}^{\text{base}} \in \arg\min_{S\in\mathbf{S}}\{c_2(\mathbf{s}, 0, \emptyset, 0) + w_b s_{n+1}\}. \tag{2.10}$$

Model 2 is very similar to Model 1 with only a difference on the valid reactions.

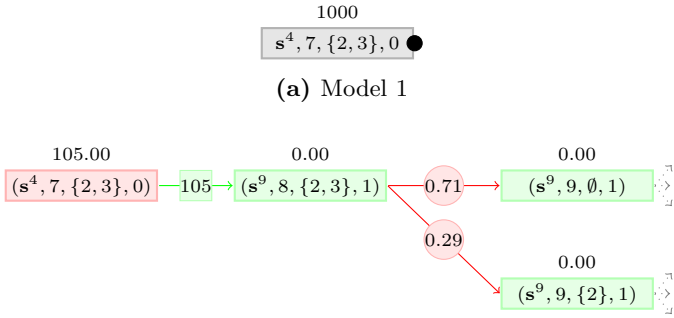
Let us define a new set $\mathbf{\Pi}_2$ as the set of all PR-policies that can be constructed using the schedules in $\mathbf{S}$ and allowing flexible transitions. Note that $\mathbf{\Pi}_1 \subseteq \mathbf{\Pi}_2$. We introduce problem P_2 as follows:

$$\mathrm{P}_2 : \min_{\Pi\in\mathbf{\Pi}_2} \sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi, l).$$

Model 2 (the recursion system (2.6)-(2.10)) optimally solves P_2. Since $\mathbf{\Pi}_1 \subseteq \mathbf{\Pi}_2$, Model 2 results in a solution that is at least as good as the solution provided by Model 1 if started from the same set of schedules. Although Model 2 searches over a much larger set of PR-policies than that in Model 1, the two models share the same worst-case computational complexity. Model 2, however, requires more computational resources (computational steps and memory requirements) than Model 1.

2.2.2.1 Graph representation of flexible transition

Figure 2.8 depicts the difference in the reaction possibilities of the two models. Model 1 provides no reacting transition to dissolve the infeasibility in state $(\mathbf{s}^4, 7, \{2,3\}, 0)$ (Figure 2.8(a)) whereas Model 2 provides a flexible transition (Figure 2.8(b)). In the flexible transition between $(\mathbf{s}^4, 7, \{2,3\}, 0)$ and $(\mathbf{s}^9, 8, \{2,3\}, 1)$, the starting times of the ongoing activities (activity 2 and activity 3) in $\mathbf{s}^4$ are $s_2^4 = 1$ and $s_3^4 = 4$ and the starting time of the ongoing activities in $\mathbf{s}^9$ are $s_2^9 = 2$ and $s_3^9 = 5$. If we choose $de = 1$, then the first condition is met. $U(\mathbf{s}^4, 7) = U(\mathbf{s}^9, 8)$



(a) Model 1

(b) Model 2

Figure 2.8: Difference between Model 1 and Model 2.

satisfies the second condition and the fact that time $7 + de = 7 + 1 = 8$ is not a decision moment in $\mathbf{s}^9$ fulfills the last condition. Thus this flexible transition is valid.

2.2.2.2 Enhancement by introducing base states

We introduce *base states* as follows. A base state which is denoted by $(\mathbf{s}, t, O)$ includes all "child" states $(\mathbf{s}, t, O, \nu)$ with $\nu \geq 0$. All states within a base state are called siblings. The decision arcs and the chance arcs that are leaving a state are very similar to those of its siblings. For instance, consider the two states $(\mathbf{s}, t, O, \nu)$ and $(\mathbf{s}, t, O, \nu + 1)$. Assume that the two chance arcs $(\mathbf{s}, t \rightarrow t', O \rightarrow O', \nu)$ with probability of occurrence 0.4 and $(\mathbf{s}, t \rightarrow t', O \rightarrow O'', \nu)$ with probability of occurrence 0.6 are leaving the state $(\mathbf{s}, t, O, \nu)$. Likewise, the two chance arcs $(\mathbf{s}, t \rightarrow t', O \rightarrow O', \nu + 1)$ with probability of occurrence 0.4 and $(\mathbf{s}, t \rightarrow t', O \rightarrow O'', \nu + 1)$ with probability of occurrence 0.6 are leaving the state $(\mathbf{s}, t, O, \nu + 1)$. Also, duplicated decision arcs leave infeasible siblings. In order to reduce computational resources needed to solve the problem, we only compute and store chance arcs and/or decision arcs for the base state and use them in computing the expected costs associated with its child states.

Moreover, we can enhance the algorithm even further for the special case where $w_{ik} = w_{i,k+1}$, $\forall i \in N, k \geq 0$. If so, then the cost of each state within a certain base state equals the cost of any of its siblings. In other

words, we have

$$c_2(\mathbf{s}, t, O, 0) = c_2(\mathbf{s}, t, O, 1) = c_2(\mathbf{s}, t, O, 2) = \dots. \tag{2.11}$$

Once a cost of a state is known, we can use its cost for all of its siblings. Therefore, the algorithm requires much less computational resources.

2.2.3 Model 3

Even though Model 2 is meant to reduce the number of deadstates, computational results show that there are still reasonable amounts of deadstates in the resulting network of Model 2, which raises serious questions on the effectiveness of Model 2. In this section, we introduce Model 3 whose network is proven to be deadstate-free.

2.2.3.1 State representation

We introduce a new state representation for Model 3: states are defined by (cu, co, de, ν). A *cu* (an abbreviation of *a project cut*) indicates the execution situation of the project at a certain decision moment and is defined by the trio $(F, O, \boldsymbol{\varrho})$ where F denotes the set of finished activities, O represents the set of ongoing activities and $\boldsymbol{\varrho}$ represents the vector of *elapsed times* of the ongoing activities. The elapsed time of an activity is the time passed since the starting of that activity. A *co* (an abbreviation of *a continuation*) indicates the starting times of the remainder of the project at a certain decision moment and is defined by the pair $(U, \boldsymbol{\rho})$ where U denotes the set of not yet started activities and $\boldsymbol{\rho}$ represents the relative starting times of the not yet started activities. If the starting time of an activity i is s_i and the continuation starts at time t, then the relative starting time of activity i for this continuation will be $s_i - t$. The value $de \geq 0$ determines the delay between the moment of the cut and the start of the continuation. Each state $(\mathbf{s}, t, O, \nu)$ in Model 1 (or Model 2) can be linearly mapped to its corresponding state (cu, co, de, ν) in Model 3. To explain how this mapping is done, we provide an example:

Example. *Figure 2.9 shows how to convert state* $(\mathbf{s}^7, 7, \{6\}, 0)$ *of Model 1 to its corresponding state* $(cu_{(\mathbf{s}^7,7,\{6\})}, co_{(\mathbf{s}^7,7)}, 0, 0)$ *of Model 3 where*

$$cu_{(\mathbf{s}^7,7,\{6\})} = (\{1,3,4\}, \{6\}, (*,*,*,*,*,*,2,*,*,*)) \text{ and}$$
$$co_{(\mathbf{s}^7,7)} = (\{2,5,7,8,9\}, (*,*,0,*,*,0,*,7,10,12)).$$

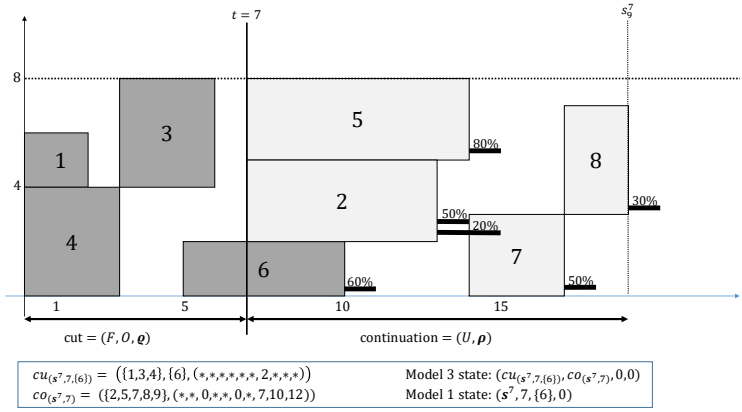


Figure 2.9: An example of a cut and a continuation.

Readers should beware that, in our examples, instead of using numerical indices for cuts and continuations (e.g., cu_1 and co_1), we use parametric combination $(\mathbf{s}, t, O)$ as the index of a cut (e.g., $cu_{(\mathbf{s}^7,7,\{6\})}$) and parametric combination $(\mathbf{s}, t)$ as the index of a continuation (e.g., $cu_{(\mathbf{s}^7,7)}$). This parametric indexing approach provides much more information to the reader than a numerical indexing approach. However, readers must note that a single unique cut (or a single unique continuation) can be represented by different parametric combinations. For instance, the cut $(\{0\},\{1,4\},(*,1,*,*,1,*,*,*,*,*))$ can be represented by both $cu_{(\mathbf{s}^1,1,\{1,4\})}$ and $cu_{(\mathbf{s}^2,1,\{1,4\})}$.

2.2.3.2 Transitions

Let $J(co,t)$ be the set of activities that start t time units after the start of the continuation ($J(co,t) = \{i \in U | \rho_i = t\}$). A state (cu, co, de, ν) is feasible if either $de > 0$ or scheduling the activities in the set $J(co,0)$ while the activities in O are ongoing is resource and precedence feasible. Remind that, if $de > 0$ then there is a positive gap between the moment of the cut and the start of the continuation and thus it is guaranteed that no activity is started until the next decision moment.

If (cu, co, de, ν) is a feasible state, then a set of chance arcs leave this state. On one hand, if $de > 0$, then there exist chance arcs $(cu \rightarrow cu', co, de \rightarrow 0, \nu)$ with $cu = (F, O, \boldsymbol{\varrho})$ and $cu' = (F', O', \boldsymbol{\varrho}')$ such that

- $F \subseteq F'$, $O' \subseteq O$ and $F \cup O = F' \cup O'$,
- if $i \in O \setminus O'$, then $de + \varrho_i \geq p_i^{\min}$,
- if $i \in O \cap O'$, then $\varrho'_i = de + \varrho_i \leq p_i^{\max}$.

The probability of occurrence of each chance arc $(cu \to cu', co, de \to 0, \nu)$ is computed as follows:

$$\pi(cu \to cu', co, de \to 0, \nu) = \prod_{i \in O \cap O'} \frac{\pi(\tilde{p}_i > de + \varrho_i)}{\pi(\tilde{p}_i > \varrho_i)} \times \prod_{i \in O \setminus O'} \left(1 - \frac{\pi(\tilde{p}_i > de + \varrho_i)}{\pi(\tilde{p}_i > \varrho_i)}\right).$$

On the other hand, if $de = 0$ in a feasible state (cu, co, de, ν), then there exists a set of chance arcs $(cu \to cu', co \to co', 0, \nu)$ with $cu = (F, O, \boldsymbol{\varrho})$, $cu' = (F', O', \boldsymbol{\varrho}')$, $co = (U, \boldsymbol{\rho})$ and $co' = (U', \boldsymbol{\rho}') = co \lhd t_{next}$ such that

- $F \subseteq F'$ and $O' \subseteq O \cup J(co, 0)$,
- if $i \in O \setminus O'$, then $t_{next} + \varrho_i \geq p_i^{\min}$,
- if $i \in O' \cap O$, then $\varrho'_i = t_{next} + \varrho_i \leq p_i^{\max}$,
- if $i \in J(co, 0) \cap O'$, then $\varrho'_i = t_{next} \leq p_i^{\max}$,
- if $i \in J(co, 0) \setminus O'$, then $t_{next} \geq p_i^{\min}$.

where $t_{next} = \min\{\rho_i > 0 | i \in U\}$ and $\lhd$ is a left-shift operator. This operator left-shifts each activity in a continuation by a constant value and removes any activity that cannot be left-shifted. The probability of occurrence of the chance arc $(cu \to cu', co \to co', 0, \nu)$ is computed as follows:

$$\pi(cu \to cu', co \to co', 0, \nu) = \prod_{i \in O \cap O'} \frac{\pi(\tilde{p}_i > t_{next} + \varrho_i)}{\pi(\tilde{p}_i > \varrho_i)} \times \prod_{i \in J(co,0) \cap O'} \pi(\tilde{p}_i > t_{next}) \times \prod_{i \in O \setminus O'} \left(1 - \frac{\pi(\tilde{p}_i > t_{next} + \varrho_i)}{\pi(\tilde{p}_i > \varrho_i)}\right) \times \prod_{i \in J(co,0) \setminus O'} (1 - \pi(\tilde{p}_i > t_{next})).$$

An example might help to clarify the above formulations:

Example. *Consider state* $(cu_{(\mathbf{s}^7,6,\{6\})}, co_{(\mathbf{s}^7,7)}, 1, 0)$*. This state is a feasible state since* $de = 1$*. There is only one chance arc leaving this state which is* $(cu_{(\mathbf{s}^7,6,\{6\})} \rightarrow cu_{(\mathbf{s}^7,7,\{6\})}, co_{(\mathbf{s}^7,7)}, 1 \rightarrow 0, \nu)$*. The probability of occurrence of this chance arc is computed as follows (note that* $O \setminus O' = \{6\} \setminus \{6\} = \emptyset$*):*

$$\pi(cu_{(\mathbf{s}^7,6,\{6\})} \rightarrow cu_{(\mathbf{s}^7,7,\{6\})}, co_{(\mathbf{s}^7,7)}, 1 \rightarrow 0, \nu) = \frac{\pi(\tilde{p}_6 > 1 + 1)}{\pi(\tilde{p}_i > 1)} = 1.$$

Now consider state $(cu_{(\mathbf{s}^7,7,\{6\})}, co_{(\mathbf{s}^7,7)}, 0, 0)$ *(see Figure 2.9). It is resource and precedence feasible to schedule the activities in* $J(co_{(\mathbf{s}^7,7)}, 0) = \{2, 5\}$ *while activity* 6 *is ongoing and thus the state is feasible. In* $\mathbf{s}^7$*, the next decision moment after* 7 *is* 14*. Since the schedule is feasible and no reaction is needed, the continuation at the next decision moment will be* $co_{(\mathbf{s}^7,7)} \lhd 7 = co_{(\mathbf{s}^7,14)}$ *(in general,* $co_{(\mathbf{s},t)} \lhd x = co_{(\mathbf{s},t+x)}$ *always holds). The detailed computations of this left shift are given below:*

$$\begin{aligned} co_{(\mathbf{s}^7,7)} \lhd 7 &= (\{2,5,7,8,9\}, (*,*,0,*,*,0,*,7,10,12)) \lhd 7 \\ &= (\{7,8,9\}, (*,*,*,*,*,*,*,0,3,5)) = co_{(\mathbf{s}^7,14)}. \end{aligned}$$

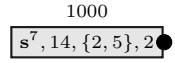
In this example, activities 2 *and* 5 *are removed because they cannot be left-shifted. The following four chance arcs leave state* $(cu_{(\mathbf{s}^7,7,\{6\})}, co_{(\mathbf{s}^7,7)}, 0, 0)$*:*

- $(cu_{(\mathbf{s}^7,7,\{6\})} \rightarrow cu_{(\mathbf{s}^7,14,\emptyset)}, co_{(\mathbf{s}^7,7)} \rightarrow co_{(\mathbf{s}^7,14)}, 0, 0)$ *with a probability of occurrence of 0.16,*
- $(cu_{(\mathbf{s}^7,7,\{6\})} \rightarrow cu_{(\mathbf{s}^7,14,\{2\})}, co_{(\mathbf{s}^7,7)} \rightarrow co_{(\mathbf{s}^7,14)}, 0, 0)$ *with a probability of occurrence of 0.04,*
- $(cu_{(\mathbf{s}^7,7,\{6\})} \rightarrow cu_{(\mathbf{s}^7,14,\{5\})}, co_{(\mathbf{s}^7,7)} \rightarrow co_{(\mathbf{s}^7,14)}, 0, 0)$ *with a probability of occurrence of 0.64,*
- $(cu_{(\mathbf{s}^7,7,\{6\})} \rightarrow cu_{(\mathbf{s}^7,14,\{2,5\})}, co_{(\mathbf{s}^7,7)} \rightarrow co_{(\mathbf{s}^7,14)}, 0, 0)$ *with a probability of occurrence of 0.16.*

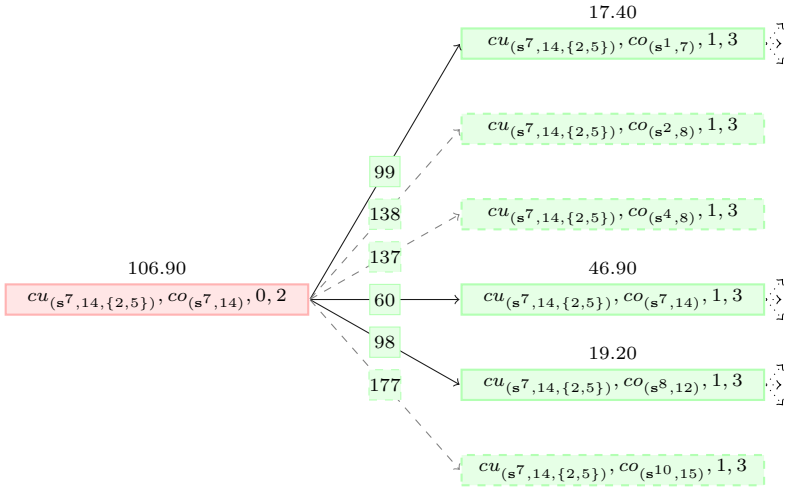
A transition (decision arc) between $(cu, co, 0, \nu)$ and $(cu, co', de', \nu+1)$ that is shown by $(cu, co \rightarrow co', 0 \rightarrow de', \nu \rightarrow \nu + 1)$ is valid only if the former state is infeasible and the latter one is feasible. The cost of this transition, which is referred to as a *cut transition*, equals:

$$w_r + \sum_{i \in U} \left(w_{i(\nu+1)} | \rho_i - (de' + \rho'_i) | \right).$$

The set of all cut transitions also includes all normal and flexible transitions. For each infeasible state $(cu, co, 0, \nu)$ (remind that any state



(a) Model 2: no reaction possibility to resolve the infeasibility in state $(\mathbf{s}^7, 14, \{2,5\}, 2)$



(b) Model 3: cut transitions that resolve the infeasibility in state $(cu_{(\mathbf{s}^7,14,\{2,5\})}, co_{(\mathbf{s}^7,14)}, 0, 2)$

Figure 2.10: The difference in reaction possibilities between Model 2 and Model 3.

(cu, co, de, ν) with $de \geq 1$ is certainly feasible), we introduce $\Gamma_3(cu, co)$ that represents the set of all pairs (co', de') for which a cut transition from $(cu, co, 0, \nu)$ to $(cu, co', de', \nu + 1)$ exists. Note that from each infeasible state $(cu, co, 0, \nu)$, there is at least one valid cut transition to state $(cu, co, 1, \nu + 1)$, and therefore it is guaranteed that the network in Model 3 is deadstate-free. Some experiments show that such a network might explode. Therefore, we enforce $de' \leq 1$ in our experiments.

Example. *Consider state* $(\mathbf{s}^7, 14, \{2,5\}, 2)$ *in Model 2 and its associated state* $(cu_{(\mathbf{s}^7,14,\{2,5\})}, co_{(\mathbf{s}^7,14)}, 0, 2)$ *in Model 3. Figure 2.10(a) depicts state* $(\mathbf{s}^7, 14, \{2,5\}, 2)$ *which is a deadstate in Model 2. While Model 2 fails to resolve the infeasibility in state* $(\mathbf{s}^7, 14, \{2,5\}, 2)$*, Model 3 proposes six different possible cut transitions to resolve the infeasibility in* $(cu_{(\mathbf{s}^7,14,\{2,5\})}, co_{(\mathbf{s}^7,14)}, 0, 2)$ *(see Figure 2.10(b)). Notice that three of*

these transitions (indicated by dashed lines) are simply too costly and therefore are dominated by the dominance rule that will be introduced in Section 2.2.3.4.

2.2.3.3 Dynamic programming recursion

First, we compute the cost until the end of the execution for each decision arc:

$$d_3(cu, co \to co', 0 \to de', \nu \to \nu+1) = w_r + \sum_{i \in U} \left(w_{i(\nu+1)} |\rho_i - (de' + \rho'_i)| \right) + c_3(cu, co', de', \nu+1). \tag{2.12}$$

Then, the cost associated with the best decision is computed as follows:

$$(co^*, de^*) \in \underset{(co', de') \in \Gamma_3(cu, co)}{\arg\min} \{ d_3(cu, co \to co', 0 \to de', \nu \to \nu+1) \} \tag{2.13}$$

$$d_3^*(cu, co, 0, \nu) = d_3(cu, co \to co^*, 0 \to de^*, \nu \to \nu+1). \tag{2.14}$$

We introduce the set $\mathcal{F}_3$, $\mathcal{I}_3$ and $\mathcal{E}_3$ as the set of all feasible states, the set of all infeasible states and the set of all endstates in Model 3, respectively. The function $c_3(cu, co, de, \nu)$ is computed as follows:

$$c_3(cu, co, de, \nu) = \sum_{(cu', co', de', \nu) \in \mathcal{F}_3 \setminus \mathcal{E}_3} \pi(cu \to cu', co \to co', de \to de', \nu) * c_3(cu', co', de', \nu) + \sum_{(cu', co', 0, \nu) \in \mathcal{I}_3} \pi(cu \to cu', co \to co', de \to 0, \nu) * d_3^*(cu', co', 0, \nu). \tag{2.15}$$

Similarly to the previous models, we need to select one of the schedules in $\mathbf{S}$ as the baseline schedule. Let us introduce $cu^0 = (\emptyset, \emptyset, ())$. We select the baseline schedule ($\mathbf{s}^{\text{base}}$) as follows:

$$\mathbf{s}^{\text{base}} \in \underset{\mathbf{s} \in \mathbf{S}}{\arg\min} \{ c_3(cu^0, co_{(\mathbf{s},0)}, 0, 0) + w_b s_{n+1} \}. \tag{2.16}$$

Let us define the set $\mathbf{\Pi}_3$ as the set of all PR-policies that can be constructed using the schedules in $\mathbf{S}$ and allowing cut transitions. Note that $\mathbf{\Pi}_1 \subseteq \mathbf{\Pi}_2 \subseteq \mathbf{\Pi}_3$. We introduce problem P_3 as follows:

$$\mathrm{P}_3 : \min_{\Pi \in \mathbf{\Pi}_3} \sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi, l).$$

The recursion system (2.12)-(2.16) optimally solves P_3 and the optimal objective value equals $c_3(cu^0, co_{(\mathbf{s}^{\text{base}},0)}, 0, 0) + w_b s^{\text{base}}_{n+1}$. Because $\mathbf{\Pi}_1 \subseteq \mathbf{\Pi}_2 \subseteq \mathbf{\Pi}_3$, Model 3 provides a solution that is at least as good as the solution provided by Model 2 or Model 1 for the same set of starting schedules.

In Model 3, we need to generate all possible continuations before starting the recursion. In order to avoid duplicates, for each continuation we make sure that the smallest relative starting time of an activity is zero, otherwise, we left-shift that continuation until this condition is fulfilled. For example, if $co = (\{1,2,3\},(2,5,7))$, then we left-shift co by two units ($co = co \lhd 2$) and therefore we will have $co = (\{1,2,3\},(0,3,5))$.

2.2.3.4 Improvement by removing dominated decision arcs

In Model 3, many cut transitions are possible. Thus, plenty of decision arcs are leaving each state. However, many of these decision arcs are very costly to choose and hence will never be part of an optimal solution. Let $\text{LB}(c_3(cu, co, de, \nu))$ be a valid lower bound for $c_3(cu, co, de, \nu)$. The following dominance rule removes those decision arcs that are too costly such that they can be dominated by at least one other decision arc.

Dominance rule 2.1. *Assume that one has partially computed the cost of an infeasible state $(cu, co, 0, \nu)$. That means he/she has already computed the expected costs until the end for some (but not all) of the decision arcs. Let $(\hat{co}, \hat{de})$ be the best decision so far for this infeasible state. For each remaining decision (co', de'), if*

$$w_r + \sum_{i \in U} \left(w_{i(\nu+1)} |\rho_i - (de' + \rho'_i)| \right) + \text{LB}(c_3(cu, co', de', \nu+1)) \geq d_3(cu, co \to \hat{co}, 0 \to \hat{de}, \nu \to \nu+1),$$

then (co', de') is certainly not the best decision for (cu, co, de, ν) and its corresponding arc $(cu, co \to co', 0 \to de', \nu \to \nu+1)$ can be eliminated.

To support the above dominance rule, it suffices to remind (2.12) and argue that the transition from $(cu, co, 0, \nu)$ to $(cu, \hat{co}, \hat{de}, \nu+1)$ is at least as good as the transition from $(cu, co, 0, \nu)$ to $(cu, co', de', \nu+1)$.

Although Dominance rule 2.1 can be a powerful dominance rule, computing a valid $\text{LB}(c_3(cu, co', de', \nu+1))$ seems to be a big obstacle. Nevertheless, we simply assume that $\text{LB}(c_3(cu, co', de', \nu+1)) = 0$. Despite being naive, a considerably large number of decision arcs are eliminated

by Dominance rule 2.1 where $\text{LB}(c_3(cu, co', de', \nu+1)) = 0$. The following example provides a better understanding of the above dominance rule.

Example. *Consider the decision arcs in Figure 2.10(b). The expected cost until the end of the execution for decision arc* $(cu_{(\mathbf{s}^7,14,\{2,5\})}, co_{(\mathbf{s}^7,14)} \rightarrow co_{(\mathbf{s}^1,7)}, 0 \rightarrow 1, 2 \rightarrow 3)$ *is* $99 + 17.40 = 116.40$ *which is already lower than the cost of the transition from state* $(cu_{(\mathbf{s}^7,14,\{2,5\})}, co_{(\mathbf{s}^7,14)}, 0, 2)$ *to state* $(cu_{(\mathbf{s}^7,14,\{2,5\})}, co_{(\mathbf{s}^2,8)}, 1, 3)$. *Thus, decision arc* $(cu_{(\mathbf{s}^7,14,\{2,5\})}, co_{(\mathbf{s}^7,14)} \rightarrow co_{(\mathbf{s}^2,8)}, 0 \rightarrow 1, 2 \rightarrow 3)$ *is dominated.*

In Section 2.4.2, we provide an approach to compute better lower bounds that are valid when certain conditions are met (*i.e.*, when non-conflict-based transitions are allowed). However, the verification of the correctness of the lower bound obtained using this approach, specially when 'non-conflict-based' transitions are not allowed, will be a topic of future research.

2.2.4 Model 4

As we already mentioned in Section 2.2.3, many very costly decision arcs exist in Model 3 and only some of them can be dominated by Dominance rule 2.1. We introduce Model 4 which is very similar to Model 3 with the only difference that the decisions are generated in a much smarter fashion.

For each combination of (cu, co, de), which represents an infeasible schedule, we compute a number of continuations using RP-SGS (note that preliminary results suggest that RP-SGS generally outperforms *robust serial SGS* (RS-SGS)) with different lists (*earliest baseline starting time* (EBST) and *lowest activity number* (LAN)) and different sets of durations ($\mathbf{p}^{50\%}$, $\mathbf{p}^{60\%}$, $\mathbf{p}^{70\%}$ and $\mathbf{p}^{80\%}$) where $\mathbf{p}^{x\%}$ is computed as follows:

$$\mathbf{p}^{x\%} = (p_0^{x\%}, p_1^{x\%}, ..., p_{n+1}^{x\%}) \text{ where}$$
$$p_i^{x\%} = \min\{p_i | p_i^{\min} \leq p_i \leq p_i^{\max}, \pi(\tilde{p}_i \leq p_i) \geq x\%\}.$$

Note that $p_i^{x\%}$ is the $\frac{x}{100}$-fractile of the distribution of $\tilde{p}_i$.

For each different choice of a reaction policy, a set of durations and a list, we possibly end up in a different continuation. To avoid duplications we ensure that the smallest relative starting time of an activity is zero, otherwise we left-shift that continuation until this condition is

The concept of non-conflict-based transitions is explained in Section 2.4.1.

fulfilled. For each combination of (cu, co), the above computations provide the set of possible reactions ($\Gamma_4(cu, co)$) with at most 8 different pairs (co', de'). Each pair (co', de') represents a *wise* transition. In comparison with $\Gamma_3(cu, co)$ (which is the set of possible reactions in Model 3), this set includes a much smaller number of pairs. However, the quality of the pairs in $\Gamma_4(cu, co)$ is, on average, meant to be much better than that of those in $\Gamma_3(cu, co)$.

The recursion system of Model 4 can be computed in the same fashion as that of Model 3 except that, in Model 4, we use $\Gamma_4(cu, co)$ instead of $\Gamma_3(cu, co)$. We compute the cost until the end of the execution for each decision arc:

$$d_4(cu, co \to co', 0 \to de', \nu \to \nu + 1) = w_r + \sum_{i \in U} \left(w_{i(\nu+1)} | \rho_i - (de' + \rho_i') | \right) + c_4(cu, co', de', \nu + 1). \tag{2.17}$$

The cost associated with the best decision is computed as follows:

$$(co^*, de^*) \in \underset{(co', de') \in \Gamma_4(cu, co)}{\arg\min} \{ d_4(cu, co \to co', 0 \to de', \nu \to \nu + 1) \} \tag{2.18}$$

$$d_4^*(cu, co, 0, \nu) = d_4(cu, co \to co^*, 0 \to de^*, \nu \to \nu + 1). \tag{2.19}$$

We introduce the set $\mathcal{F}_4$, $\mathcal{I}_4$ and $\mathcal{E}_4$ as the set of all feasible states, the set of all infeasible states and the set of all endstates in Model 4, respectively. The function $c_4(cu, co, de, \nu)$ is computed as follows:

$$c_4(cu, co, de, \nu) = \sum_{(cu', co', de', \nu) \in \mathcal{F}_4 \setminus \mathcal{E}_4} \pi(cu \to cu', co \to co', de \to de', \nu) * c_4(cu', co', de', \nu) + \sum_{(cu', co', 0, \nu) \in \mathcal{I}_4} \pi(cu \to cu', co \to co', de \to 0, \nu) * d_4^*(cu', co', 0, \nu). \tag{2.20}$$

We select the baseline schedule ($\mathbf{s}^{\text{base}}$) as follows:

$$\mathbf{s}^{\text{base}} \in \underset{\mathbf{s} \in \mathbf{S}}{\arg\min} \{ c_4(cu^0, co_{(\mathbf{s},0)}, 0, 0) + w_b s_{n+1} \}. \tag{2.21}$$

Let us define a new set $\mathbf{\Pi}_4$ as the set of all PR-policies that can be constructed using the baseline schedules in $\mathbf{s}$ and allowing wise transitions. It should be noted that $\mathbf{\Pi}_4 \not\subseteq \mathbf{\Pi}_3$. We introduce problem P_4 as follows:

$$\mathrm{P}_4 : \min_{\Pi \in \mathbf{\Pi}_4} \sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi, l).$$

Model 4 (the recursion system (2.17)-(2.21)) optimally solves P_4.

2.3 Computational results

All models (Models 1-4) have been implemented in Visual C++ 2010. All computational results, unless stated otherwise, were obtained on a laptop Dell Latitude with 2.6 GHz Core(TM) i7-3720QM processor, 8GB of RAM and running under Windows 10.

2.3.1 Instance generation

Our models are tested on a set of 48 instances that are composed of PSPLIB instances. Only instances with 30 non-dummy activities are considered in this experiment. PSPLIB is a class of instances for the deterministic RCPSP (Kolisch and Sprecher, 1997), thus they need to be modified to suit our problem. The following modifications are applied on this set of instances: the activity duration $\tilde{p}_i$ for non-dummy activity i follows a discretized beta distribution with shape parameters 2 and 5 that is mapped over the interval $[0.75\hat{p}_i, 1.625\hat{p}_i]$ where $\hat{p}_i$ is the duration of activity i that is given in the original instance.

The base activity weights w_{j0} for $j \in \{1, 2, ..., n\}$ are obtained from a discrete triangular distribution with $Pr(w_j = q) = (21 - 2q)\%$ for $q \in \{1, 2...10\}$. This distribution results in a higher probability for low weights. The average weight w_{avg} then equals 3.85 which is used to calculate the weight of the dummy end activity $w_{n+1,0} = 38 \approx 10w_{\text{avg}}$. The non-base activity weights are computed as follows: $w_{jk} = \lambda^k w_{j0}; k = 1, \ldots, 30$. The value of λ is chosen from $\{1, 1.1, 1.2\}$.

In order to reduce the number of experiments, we only consider the instances from the set `J30` of PSPLIB with the following filename syntax: `J30X_1` (`X` $= 1, ..., 48$).

2.3.2 Measures of stability and robustness

In this section, we introduce and discuss two measures that are used as comparison measures:

- Combined cost (CC): the expected proactive and reactive cost for each PR-policy Π which is computed as follows:

$$\sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi, l).$$

- Recovery robustness (RR): the probability of not having a deadend which is computed by:

$$\sum_{l=1}^{|\mathfrak{P}|} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l)(1 - \gamma_{\Pi,l}).$$

2.3.3 Results for our proposed models

In this section, we discuss the results of our four proposed models. For each model, we report the average CPU time (in seconds) required to solve the problem (CPU), the average combined cost (CC), the average recovery robustness (RR) and the average number of states (#states). For Models 1 and 2 we also report the average number of chance arcs (#c arcs) and the average number of decision arcs (#d arcs), whereas for Models 3 and 4 we report the average number of cuts (#cuts) and the average number of continuations (#cons).

2.3.3.1 Model 1

We run Model 1 on the set of instances explained above and over the nine different sets of schedules. These nine different sets are constructed using three different initial sets of schedules and three different values for κ_1. For this experiment, we choose $w_b = 50$, $\lambda = 1$, $w_r = 100$ and $M = 1000$.

Table 2.3 depicts an overview of the results obtained for Model 1 on the nine different sets of schedules. Note that in order to reduce the space required to store all arcs, states with the same $\mathbf{s}$, t and O are grouped into bigger states (called base state) and then arcs are only added among base states. Therefore, it might be possible that the number of states is greater than the number of chance arcs and/or the number of decision arcs.

There are improving trends in both combined cost and recovery robustness by increasing the size of the set of schedules. The only exception is when we generate schedules using DET. In this case, the combined cost

$\mathbf{S}^{\text{init}}$	κ_1	CPU	CC	RR	#states	#c arcs	#d arcs
DET	500	1.06	4034.94	0.07	236561	105149	260149
	1000	2.97	4040.95	0.09	519700	217163	1027909
	2000	9.24	4038.69	0.15	1139451	448983	4073017
STC	500	20.96	3946.84	0.42	217502	104098	219300
	1000	24.17	3934.67	0.48	485080	213790	882217
	2000	29.37	3919.82	0.53	1064437	440083	3563275
SMP	500	5.69	4011.25	0.38	273222	106733	223245
	1000	7.09	3988.70	0.41	595881	220948	886837
	2000	13.38	3957.46	0.47	1296020	456841	3533487

Table 2.3: Summary of the results for Model 1.

is fluctuating while the recovery robustness is improved by increasing the size of the set of schedules. This behavior is considered to be strange because more schedules results in more reaction possibilities and smaller numbers of deadends and thus it is expected that the combined cost is decreased. This strange behavior is possible when the expected cost imposed by the added reaction possibilities (when we use the larger set of schedules) is larger than the expected cost imposed by the deadends (when we use the smaller set of schedules). For example, when we use DET to generate 500 schedules, recovery robustness is around 0.07. When we use DET to generate 1000 schedules, recovery robustness is around 0.09 which means that the probability of having a deadend when we generate 1000 schedules is around 2% less than that when we only generate 500 schedules. This means that in 2% of the cases, instead of paying M which is the cost of the deadends, we must pay the costs of reactions. So, whenever the expected cost of these reactions is higher than the expected cost of deadends (*i.e.*, $1000 \times 0.02 = 20$), this strange behavior occurs.

Also, we notice that both combined cost and recovery robustness significantly improve, while CPU time increases, by using STC or SMP instead of DET as the initial set of schedules. On average, it takes around 20 seconds to generate 13 schedules in the set STC and around 4.5 seconds to generate 50 schedules in SMP. Finally, we observe that as we increase the number of schedules, the number of states and the number of chance arcs are linearly increased whereas the increase in the number of decision arcs is rather quadratic.

$\mathbf{S}^{\text{init}}$	κ_1	CPU	CC	RR	#states	#c arcs	#d arcs
DET	500	0.87	3739.34	0.80	61347	135441	1123350
	1000	2.38	3686.56	0.88	126631	284889	4606344
	2000	8.76	3645.05	0.91	260473	596044	18841063
STC	500	20.52	3696.72	0.86	64640	143110	1058972
	1000	22.05	3641.85	0.91	133056	299560	4317168
	2000	27.19	3607.48	0.94	273195	625034	17593573
SMP	500	5.42	3696.85	0.89	71156	167705	987363
	1000	6.67	3649.07	0.92	146259	349139	3974585
	2000	11.39	3614.29	0.94	300199	722934	15943784

Table 2.4: Summary of the results for Model 2.

2.3.3.2 Model 2

The summary of the results obtained for Model 2 are reported in Table 2.4. For this experiment, similarly to Model 1, we choose $w_b = 50$, $\lambda = 1$, $w_r = 100$ and $M = 1000$. Obviously, Model 2 outperforms Model 1 in both combined cost and recovery robustness. Although Model 2 runs faster than Model 1 when $\lambda = 1$ which is mainly a result of the enhancement techniques explained in Section 2.2.2.2, it needs more computational resources (both computation steps and memory requirements) than those needed for Model 1 to solve the problem. Similarly to Model 1, increasing the size of the set of schedules and/or switching from DET to either STC or SMP result in both better combined cost and better recovery robustness. We notice that in spite of a huge improvement in recovery robustness (solutions with on average 80-95% recovery robustness), we fail to find fully recovery-robust solutions (solutions with 100% recovery robustness) for any of the instances.

The numbers of states in Model 2 are much smaller than those of Model 1 because in Model 2 we use base states to reduce the computational resources as explained in Section 2.2.2.2. However, if $\lambda \neq 1$, then the numbers of states in Model 2 would have been larger than those in Model 1. Unlike the numbers of states, the numbers of arcs in Model 2 are larger than the numbers of arcs in Model 1. Even though the numbers of chance arcs haven't increased much in comparison with Model 1, the increase in the numbers of decision arcs is quite significant.

$\mathbf{S}^{\text{init}}$	κ_1	CPU	CC	RR	#states	#cuts	#cons
DET	500	177.21	3589.07	1.00	2740331	12337	5278
	1000	736.04	3576.22	1.00	5781750	14886	10040
	2000	-	-	-	-	-	-
STC	500	244.45	3562.03	1.00	3663497	17378	6179
	1000	1029.93	3549.13	1.00	7978517	21270	11643
	2000	-	-	-	-	-	-
SMP	500	180.58	3566.60	1.00	3887518	30545	6016
	1000	790.71	3555.25	1.00	8587042	38190	11483
	2000	-	-	-	-	-	-

Table 2.5: Summary of the results for Model 3.

2.3.3.3 Model 3

The summary of the results obtained for Model 3 are reported in Table 2.5. For this experiment, similarly to Model 1 and Model 2, we choose $w_b = 50$, $\lambda = 1$, $w_r = 100$ and $M = 1000$. In Model 3, by definition all solutions must be fully recovery robust, which is also confirmed in the results. Apart from having fully recoverable robust solutions, Model 3 also provides much cheaper solutions. The improvement on the combined cost is significant if we choose Model 3 instead of Model 2. But these improvements come at a cost. The computational resources needed to solve Model 3 are much higher than those needed for Model 2. For instance, the number of states required to solve Model 3 over the set of 500 schedules constructed by $\mathbf{S}^{init}$ = DET for the given setting is around 45 times the number of states required to solve Model 2 over the same set of schedules. Also, the CPU time required to solve Model 3 is much higher than that required to solve Model 2. When $\kappa_1 = 2000$, often more computational resources are needed than available and thus many instances cannot be solved.

2.3.3.4 Model 4

The summary of the results obtained for Model 4 are reported in Table 2.6. For this experiment, similarly to Model 1, Model 2 and Model 3, we choose $w_b = 50$, $\lambda = 1$, $w_r = 100$ and $M = 1000$. Model 4 requires a huge amount of memory because the number of cuts and the number of continuations are much larger than those in Model 3. Therefore, we stop Model 4 if we reach a certain number of states. In this experiment, we stop Model 4 if we reach 40 million states. In our implementation

$\mathbf{S}^{\text{init}}$	κ_1	#solved	CPU	CC	#states	#cuts	#cons
DET	500	43	167.59	3580.87	10158884	35051	500063
	1000	43	212.46	3572.01	12566521	35738	597405
	2000	39	268.05	3565.42	15700267	36080	734225
STC	500	41	190.16	3552.42	10386517	34090	518785
	1000	41	220.60	3544.55	12257837	34432	606361
	2000	40	273.30	3539.53	15149351	34849	743418
SMP	500	36	340.14	3558.44	19269453	84722	901571
	1000	32	380.09	3554.22	22111245	85310	1013003
	2000	28	437.62	3549.38	25398410	85863	1158909

Table 2.6: Summary of the results for Model 4.

Model 4 first computes $c_4(cu^0, co^{\mathbf{s}^1}, 0, 0)$, then $c_4(cu^0, co^{\mathbf{s}^2}, 0, 0)$, and continues so until it computes $c_4(cu^0, co^{\mathbf{s}^{|\mathbf{S}|}}, 0, 0)$ as the last state. For a certain instance, if before reaching the limit of 40 million states, Model 4 computes $c_4(cu^0, co^{\mathbf{s}^{|\mathbf{S}|}}, 0, 0)$, then that instance is consider as *solved* otherwise Model 4 outputs the best $c_4(cu^0, co^{\mathbf{s}}, 0, 0) + w_b s_{n+1}$ among those computed so far.

In Model 4, similarly to Model 3, we aim to find solutions with full recovery robustness, which is also confirmed in the results. Model 4 also provides much cheaper solutions in comparison with all other models. The improvement on the combined cost is significant if we choose Model 4 instead of any other model. But these improvements come with a large increase of the number of states, the number of cuts and the number of continuations. For instance, the number of states required to solve Model 4 over the set of 500 schedules is around 5 times the number of states required to solve Model 3 over the same set of schedules. However, as we increase the number of schedules from 500 to 1000, the numbers of states in Model 3 are approximately increased by a factor of more than 2 whereas the number of states in Model 4 are approximately increased by a factor of only 1.2. The average CPU time required to solve Model 4 is less than that of Model 3 because, unlike Model 3 which has many dominated decision arcs, Model 4 is not likely to have many dominated arcs.

2.3.4 Comparison with a conventional proactive and reactive method

We compare Models 2-4 with a combination of conventional proactive and reactive approaches (the reason that we leave out Model 1 in these comparisons is that it performs much worse than the other two models). As a proactive approach we choose a modified version of the STC method (M-STC) which is introduced in Section 2.3.4.1 and as a reactive approach we choose RP-SGS where EBST is the scheme's priority rule. We refer to this combination as CONV.

2.3.4.1 Modified-STC (M-STC)

The original STC method introduced by Van de Vonder et al. (2008) produces a reasonably robust baseline schedule whose makespan is not larger than $\alpha \times makespan$ where $makespan$ is the project completion time of the optimal schedule for the deterministic RCPSP. Remember that the notation α is borrowed from the notation system in Van de Vonder et al. (2008) and as such must not be confused with $(1 - \alpha)$ that denotes the confidence level in Chapter 4. Given different values for α, STC produces different schedules, some being very tight ($1 \leq \alpha < 1.15$), some being very loose ($\alpha \geq 1.3$) and some being neither tight nor loose ($1.15 \leq \alpha < 1.3$). Depending on what values we choose for w_b, w_r and λ, it might be beneficial to have a loose or tight baseline schedule. It is indeed very difficult to incorporate w_r and λ in the computation of the stability cost exploited in the STC method, but it is possible to incorporate w_b in the computation of this cost.

The function $stc(i)$ represents the stability cost of activity i in the original method introduced by Van de Vonder et al. (2008). Let us introduce the function $\hat{stc}(i)$ as the Modified stability cost of activity i. We compute $\hat{stc}(i)$ as follows:

$$\hat{stc}(i) = stc(i) \text{ if } i \neq n+1 \text{ and}$$
$$\hat{stc}(n+1) = stc(n+1) + w_b s_{n+1}.$$

All steps in our modified STC method are exactly the same as in the original method except that we use $\sum_{i \in N} \hat{stc}(i)$ instead of $\sum_{i \in N} stc(i)$ as the total stability cost.

2.3.4.2 Modified Model 2 (M-Model 2)

Model 2 is not suitable to be compared with any conventional proactive and reactive solution, since its optimal policies may include deadends. In the modified version of Model 2, once we reach a deadstate, instead of incurring a cost of M, we run a very limited simulation to compute an estimated cost of reactions for the remainder of the execution of the project. Each simulation only consists of 10 randomly chosen realizations. During the simulation, all infeasiblities are resolved by RP-SGS where EBST is the scheme's priority rule.

2.3.4.3 M-Model 2 vs. CONV

We compare CONV and M-Model 2 where $\mathbf{S}^{init} = \text{SMP}$ and $\kappa_1 = 1000$. We apply the following experiment: we run both methods for each combination of parameters $(\lambda, w_b, w_r, \alpha)$ where $\lambda = 1, 1.1$ or 1.2, $w_b = 25$ or 50, $w_r = 0, 50, 100$ or 200 and $\alpha = 1.1, 1.2$ or 1.3. To incorporate α in M-Model 2, we limit the choice of the baseline schedule and replace (2.10) by

$$\mathbf{s}^{\text{base}} \in \arg\min_{\mathbf{s}\in\mathbf{S}_\alpha}\{c_2(\mathbf{s}, 0, \emptyset, 0) + w_b s_{n+1}\}. \tag{2.22}$$

where $\mathbf{S}_\alpha = \{\mathbf{s} \in \mathbf{S} | s_{n+1} \leq \alpha \times makespan\}$

Table 2.7 reports the outcome of this experiment. We notice that M-Model 2 performs worse than CONV when $w_r = 0$. However, we observe that by increasing the cost w_r, M-Model 2 tends to outperform CONV. Similarly, by increasing λ, the average percent deviation of M-Model 2 from CONV is also decreased.

2.3.4.4 Model 3 vs. CONV

We compare CONV and Model 3 where $\mathbf{S}^{init} = \text{SMP}$ and $\kappa_1 = 1000$. We apply a similar experiment to the one in Section 2.3.4.3 except that we fix $\lambda = 1$. To incorporate α in Model 3, we limit the choice of the baseline schedule and replace (2.16) by

$$\mathbf{s}^{\text{base}} \in \arg\min_{\mathbf{s}\in\mathbf{S}_\alpha}\{c_3(cu^0, co^{\mathbf{s}}, 0, 0) + w_b s_{n+1}\}. \tag{2.23}$$

Table 2.8 reports the outcome of this experiment. In terms of average percent deviation from CONV, Model 3 clearly outperforms M-Model 2. We notice that Model 3 also performs worse than CONV when $w_r = 0$ and

w_r	λ	$w_b = 25$			$w_b = 50$		
		$\alpha = 1.1$	$\alpha = 1.2$	$\alpha = 1.3$	$\alpha = 1.1$	$\alpha = 1.2$	$\alpha = 1.3$
0	1	4.84	3.79	3.79	2.70	2.62	2.62
	1.1	4.69	2.97	2.97	2.66	2.47	2.47
	1.2	2.93	0.27	0.27	1.24	0.75	0.75
50	1	0.66	-1.77	-1.77	0.74	0.51	0.51
	1.1	0.26	-3.08	-3.08	0.23	-0.27	-0.27
	1.2	-1.48	-5.82	-5.83	-1.40	-2.33	-2.33
100	1	-3.85	-7.98	-8.01	-2.06	-2.71	-2.71
	1.1	-4.17	-9.15	-9.20	-2.62	-3.66	-3.66
	1.2	-5.77	-11.56	-11.64	-4.19	-5.73	-5.73
200	1	-12.20	-18.40	-18.70	-7.93	-9.69	-9.70
	1.1	-12.25	-19.18	-19.59	-8.34	-10.60	-10.61
	1.2	-13.37	-20.88	-21.36	-9.65	-12.42	-12.44

Table 2.7: Average percent deviation of M-Model 2 from CONV for different values of parameters λ, w_b, w_r and α.

w_r	$w_b = 25$			$w_b = 50$		
	$\alpha = 1.1$	$\alpha = 1.2$	$\alpha = 1.3$	$\alpha = 1.1$	$\alpha = 1.2$	$\alpha = 1.3$
0	2.13	1.80	1.80	0.41	0.36	0.36
50	-1.08	-2.90	-2.90	-0.34	-0.51	-0.51
100	-5.24	-9.36	-9.42	-2.84	-3.42	-3.42
200	-13.60	-20.26	-20.71	-8.46	-10.14	-10.15

Table 2.8: Average percent deviation of Model 3 from CONV for different values of parameters w_b, w_r and α and fixed value of parameter $\lambda = 1$.

performs better than CONV when $w_r > 0$. We must acknowledge that although Model 3 outperforms M-Model 2 in all settings and CONV in most of the settings, it also requires much more computational resources than both M-Model 2 and CONV, specially when $\lambda \neq 1$.

2.3.4.5 Model 4 vs. CONV

Similarly to the previous comparisons, we compare CONV and Model 4 where $\mathbf{S}^{init} = \text{SMP}$ and $\kappa_1 = 1000$. We stop the algorithm of Model 4 if it reaches 200 million states (which requires around 25 GB of memory to store the states and the arcs). Due to a lack of memory in the laptop ma-

w_r	$w_b = 25$			$w_b = 50$		
	$\alpha = 1.1$	$\alpha = 1.2$	$\alpha = 1.3$	$\alpha = 1.1$	$\alpha = 1.2$	$\alpha = 1.3$
0	0.33	0.19	0.19	-0.47	-0.52	-0.52
50	-2.36	-3.96	-3.96	-0.91	-1.08	-1.08
100	-5.85	-10.11	-10.20	-3.11	-3.77	-3.77
200	-13.28	-20.47	-21.05	-8.52	-10.73	-10.75

Table 2.9: Average percent deviation of Model 4 from CONV for different values of parameters w_b, w_r and α and fixed value of parameter $\lambda = 1$.

chine, the computational results in this section were obtained on a server machine with Intel Xeon CPU E5-2699 2.3 GHz (2 processors), 256GB of RAM and running under Windows server 2012 R2. To incorporate α in Model 4, we limit the choice of the baseline schedule and replace (2.21) by

$$\mathbf{s}^{\text{base}} \in \arg\min_{\mathbf{s}\in\mathbf{S}_\alpha}\{c_4(cu^0, co^{\mathbf{s}}, 0, 0) + w_b s_{n+1}\}. \tag{2.24}$$

Table 2.9 reports the outcome of this experiment. In terms of average percent deviation from CONV, Model 4 clearly outperforms both M-Model 2 and Model 3. Model 4 performs slightly worse than CONV when $w_r = 0$ and $w_b = 25$ and performs better elsewhere, specially when $w_r \geq 50$. From a theoretical point of view, Model 4 must outperform CONV if $\mathbf{S}$ contains the baseline schedules that are used in CONV (which is the case when $\mathbf{S}^{init} = \text{STC}$). However, in this experiment, we construct $\mathbf{S}^{init}$ using our sampling approach (SMP) and thus the fact that CONV outperforms Model 4 when $w_r = 0$ and $w_b = 25$ can be justified.

2.4 Discussion

2.4.1 Non-conflict-based PR-policies

It might be wrongfully perceived that reactions are only necessary when conflicts occur. On the contrary, we find that reactions can be beneficial even if no conflict occurs. Upon entering a feasible state, some information becomes available that can be used to predict a conflict in the (near) future. In such a situation, it might be less costly to already transit to another much more stable schedule before the predicted conflict actually

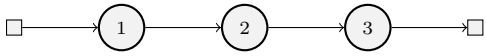


Figure 2.11: The precedence graph for the instance of the counterexample.

occurs. Such a transition is called a *non-conflict-based* transition. A PR-policy that contain at least one non-conflict-based transition is called a *non-conflict-based* PR-policy. To support our finding, we provide a counterexample where a *non-conflict-based* PR-policy provides a better solution than the optimal solution to P. Notice that, in our research, $\mathbf{\Pi}$ only contains conflict-based PR-policies.

Consider the following instance of P. We are given a project with three non-dummy activities. The set of activities is given as $N = \{0, 1, 2, 3, 4\}$ where activities 0 and 4 are dummy start and dummy end activities, respectively. We only have one resource type with availability one. Each activity needs exactly one resource unit to be executed. Figure 2.11 shows the precedence relations among the activities of this instance. The set $\mathfrak{P} = \{\mathbf{p}^1, \mathbf{p}^2, \mathbf{p}^3, \mathbf{p}^4\}$ for this instance contains only four realizations that are given as follows:

$$\begin{aligned} \mathbf{p}^1 &= (0, 1, 1, 2, 0) & \pi(\tilde{\mathbf{p}} = \mathbf{p}^1) &= 0.21 \\ \mathbf{p}^2 &= (0, 1, 2, 2, 0) & \pi(\tilde{\mathbf{p}} = \mathbf{p}^2) &= 0.09 \\ \mathbf{p}^3 &= (0, 2, 1, 2, 0) & \pi(\tilde{\mathbf{p}} = \mathbf{p}^3) &= 0.49 \\ \mathbf{p}^4 &= (0, 2, 2, 2, 0) & \pi(\tilde{\mathbf{p}} = \mathbf{p}^4) &= 0.21 \end{aligned}$$

For this counterexample, we set the parameters $w_{0,k} = w_{1,k} = w_{2,k} = 0$, $w_{3,k} = 20$, $w_{4,k} = 40$ for each $k \geq 0$ and let $w_b = 50$, $w_r = 20$, $M = 1000$.

There is a set $\mathbf{S}^{dom}$ of schedules with which we can construct every dominant PR-policy. For this very small instance, $\mathbf{S}^{dom}$ is very small and only contains the following four schedules:

$$\begin{aligned} \mathbf{s}^1 &= (0, 0, 1, 2, 4), & \mathbf{s}^2 &= (0, 0, 1, 3, 5), \\ \mathbf{s}^3 &= (0, 0, 2, 3, 5) \text{ and} & \mathbf{s}^4 &= (0, 0, 2, 4, 6). \end{aligned}$$

The conflict-based PR-policy $\Pi^* \in \mathbf{\Pi}$, which is given below, happens to be the optimal PR-policy for this instance of P.

$$\Pi^* : \begin{cases} \Phi_{\Pi^*,1} : \mathbf{s}^3 & \gamma_{\Pi^*,1} = 0, \quad f(\Pi^*, 1) = 250 \\ \Phi_{\Pi^*,2} : \mathbf{s}^3 \xrightarrow{t=3} \mathbf{s}^4 & \gamma_{\Pi^*,2} = 0, \quad f(\Pi^*, 2) = 250 + 80 \\ \Phi_{\Pi^*,3} : \mathbf{s}^3 & \gamma_{\Pi^*,3} = 0, \quad f(\Pi^*, 3) = 250 \\ \Phi_{\Pi^*,4} : \mathbf{s}^3 \xrightarrow{t=3} \mathbf{s}^4 & \gamma_{\Pi^*,4} = 0, \quad f(\Pi^*, 4) = 250 + 80 \end{cases}$$

$$\sum_{l=1}^{4} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi^*, l) = (0.21 + 0.49) \times 250 + (0.09 + 0.21) \times 330 = 274.$$

Now consider the following non-conflict-based PR-policy Π':

$$\Pi' : \begin{cases} \Phi_{\Pi',1} : \mathbf{s}^3 \xrightarrow{t=1} \mathbf{s}^2 & \gamma_{\Pi',1} = 0, \quad f(\Pi', 1) = 250 + 20 \\ \Phi_{\Pi',2} : \mathbf{s}^3 \xrightarrow{t=1} \mathbf{s}^2 & \gamma_{\Pi',2} = 0, \quad f(\Pi', 2) = 250 + 20 \\ \Phi_{\Pi',3} : \mathbf{s}^3 & \gamma_{\Pi',3} = 0, \quad f(\Pi', 3) = 250 \\ \Phi_{\Pi',4} : \mathbf{s}^3 \xrightarrow{t=3} \mathbf{s}^4 & \gamma_{\Pi^*,4} = 0, \quad f(\Pi', 4) = 250 + 80 \end{cases}$$

$$\begin{aligned} \sum_{l=1}^{4} \pi(\tilde{\mathbf{p}} = \mathbf{p}^l) f(\Pi', l) &= 0.49 \times 250 + (0.09 + 0.21) \times 270 + 0.21 \times 330 \\ &= 272.8. \end{aligned}$$

We conclude that there might be non-conflict-based PR-policies with a smaller combined cost than the best conflict-based PR-policy.

2.4.2 A possible tight lower bound

Similar models to Models 1-4 can be introduced to find the optimal non-conflict-based PR-policy. One may wonder whether or not, when considering non-conflict-based PR-policies, we can achieve a better lower bound for the expected cost until the end of execution upon leaving each feasible state. In the following paragraph, we introduce an approach in which we can efficiently compute better quality lower bounds that can be used in such models.

Consider Model 3 and continuation $cu = (F, O, \boldsymbol{\varrho})$ and let $F(cu) = F$. Based on the following conjecture, we might be able to assume that

$$\mathrm{LB}(c_3(cu, co', de', \nu + 1)) = c_3(cu'', co', de', \nu + 1)$$

where $cu'' = (F(cu), \emptyset, ())$.

Conjecture 2.1. *In an alternative model where non-conflict-based transitions are also allowed, the value* $c_3(cu'', co', de', \nu + 1)$ *where* $cu'' = (F(cu), \emptyset, ())$ *is a valid lower bound for the expected cost until the end of execution upon leaving any given feasible state* $(cu, co', de', \nu + 1)$.

2.5 Summary and future research

In this chapter, we introduced a proactive and reactive resource-constrained project scheduling problem with duration uncertainty. The optimal solution of the problem is a PR-policy with the minimum expected combined cost. This expected combined cost is a combination of an expected cost of the baseline schedule, an expected cost of a series of reactions and an expected cost of having no reaction possibility. We propose four different Markov decision process models which can solve the problem over four different classes of policies. The computational results state that Model 2, Model 3 and Model 4 perform better than the conventional method introduced in Section 2.3 when the fixed reaction cost is large ($w_r \geq 100$). When $w_r = 50$, Model 3 and Model 4 generally outperform the conventional method whereas Model 2 performs competitively against the conventional method. When $w_r = 0$, the conventional method outperforms Model 2 and Model 3 and performs competitively against Model 4.

In future research, one might consider finding solutions with even lower combined cost. Finding a diverse set of schedules which provides good reaction possibilities among schedules is likely to reduce the combined cost of the optimal solution in our models. Therefore, one future research direction is to search for a wisely selected set of schedules that provides both diversity and similarity among schedules. As a second future research direction, finding new dominance rules to eliminate dominated reaction possibilities might help reducing the computational resources needed to solve our models. Apart from the two above mentioned directions, it seems that investigating non-conflict-based PR-policies in much more detail is very important and rewarding. Finally, we suggest interested researchers to study the problem when the activity durations are dependent.

Chapter 3

The proactive and reactive resource-constrained project scheduling problem: The crucial role of buffer-based reactions

A goal without a plan is just a wish.

- Antoine de Saint-Exupery

In the previous chapter, we introduced the proactive and reactive resource-constrained project scheduling problem (PR-RCPSP) in which the goal is to find an optimal proactive and reactive policy (PR-policy). In the first section of this chapter, we aim at understanding the importance of certain classes of reactions (*i.e.*, the class of selection-based reactions and the class of buffer-based reactions) in constructing an optimal PR-policy. In the second section, we take advantage of the insights provided in the first section and propose an iterative schedule refinement technique that outputs a set of schedules whose PR-policy is less expensive than that of any of the methods introduced in the previous chapter.

The results presented in this chapter also appears in a FEB Research Report at KU Leuven (Davari and Demeulemeester, 2017).

3.1 Two important classes of reactions

In this section, we study two classes of reactions, namely the class of *selection-based* reactions and the class of *buffer-based* reactions. Both selection-based reactions and buffer-based reactions are based on *sufficient selections*. We first define what is a sufficient selection in Section 3.1.1, and then introduce the concepts of selection-based reactions and buffer-based reactions in Sections 3.1.2 and 3.1.3, respectively.

Our initial motivation to introduce these two classes of reactions was the fact that we expected that those reactions among schedules which are resulting from the same sufficient selections are often selected in the optimal PR-policy. Therefore, the ultimate objective of this section is to see what percentage of reactions in an optimal PR-policy are selection-based and what percentage of reactions in an optimal PR-policy are buffer-based.

3.1.1 Sufficient selection

A set FS of activities is a *forbidden set* if $E \cap (FS \times FS) = \emptyset$ and $\exists k \in \mathcal{R} : \sum_{i \in FS} r_{ik} > R_k$. A forbidden sets FS is *minimal* if for every $i \in FS$, the set $FS \setminus \{i\}$ is not a forbidden set. We define $\mathscr{F}(\cdot)$ as the set of all minimal forbidden sets with $\cdot$ being a partial order among activities. For instance, given the set E of precedence relations among activities, $\mathscr{F}(E)$ is the set of minimal forbidden sets. The concept of forbidden sets was first introduced by Igelmund and Radermacher (1983a).

One may use extra resource arcs to eliminate all minimal forbidden sets. Similar ideas have been implemented to address resource incompatibilities in both deterministic and stochastic resource-constrained project scheduling problems (*e.g.*, Alvarez-Valdes and Tamarit, 1993; Artigues et al., 2013; Leus, 2011a,b). Let us define $X \subset N \times N \setminus T(E)$ as a set of pairs where each pair represents a resource arc. We assume that X is a *strict partial order* (abbreviated by sp-order) on N (*i.e.*, irreflexible and transitive). For each $(i, j) \in X$, the completion time of activity i must be smaller than or equal to the starting time of activity j. We follow Leus (2011b) to call X a *selection*.

Definition 3.1 (Sufficient selection)**.** *A selection X is called sufficient if and only if $G(N, E \cup X)$ is acyclic and $\mathscr{F}(E \cup X) = \emptyset$.*

Example. *Consider the instance provided in Section 2.1.3. For the reader's convenience, we repeat this instance's data in Figure 3.1 and Tables 3.1*

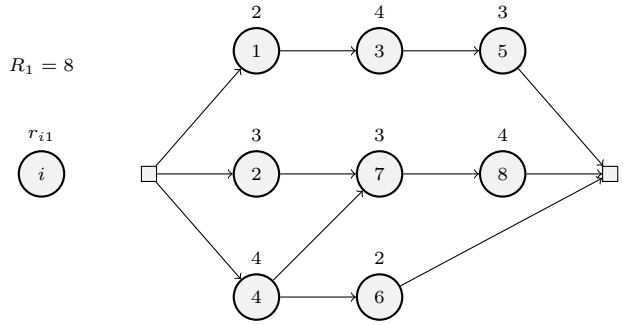


Figure 3.1: A copy of Figure 1.1.

	$\hat{p}_i$	$\pi(\tilde{p}_i = \hat{p}_i + \epsilon)$			$w_{i,0}$
		$\epsilon = -1$	$\epsilon = 0$	$\epsilon = +1$	
$\tilde{p}_0$	0	0	1	0	-
$\tilde{p}_1$	2	0.4	**0.4**	0.2	4
$\tilde{p}_2$	7	0.3	0.5	**0.2**	4
$\tilde{p}_3$	3	0	**0.6**	0.4	7
$\tilde{p}_4$	4	0.1	0.5	**0.4**	1
$\tilde{p}_5$	8	**0.2**	0.8	0	4
$\tilde{p}_6$	6	**0.4**	0.6	0	1
$\tilde{p}_7$	4	0.5	**0.5**	0	1
$\tilde{p}_8$	2	0	**0.7**	0.3	1
$\tilde{p}_9$	0	0	1	0	38

Table 3.1: A copy of Table 2.1.

and 3.2. Note that this instance is used in all examples in this section. The AON representation of the precedence relations among the activities for this instance is shown in Figure 3.1. For this instance, the set of minimal forbidden sets is

$$\mathscr{F}(E) = \{\{1,2,4\},\{2,3,4\},\{2,3,6\},\{2,4,5\},\{3,6,7\},\{3,6,8\},\{5,6,8\}\}.$$

Consider selection $X_1 = \{(4,2),(3,6),(6,8)\}$*. Let us include these arcs into the precedence network. The network in Figure 3.2 results. We notice that* X_1 *is a sufficient selection because it suffices to eliminate all minimal forbidden sets (i.e.,* $\mathscr{F}(E \cup X_1) = \emptyset$*).*

	$\mathbf{s}^k$									
	$\mathbf{s}^1$	$\mathbf{s}^2$	$\mathbf{s}^3$	$\mathbf{s}^4$	$\mathbf{s}^5$	$\mathbf{s}^6$	$\mathbf{s}^7$	$\mathbf{s}^8$	$\mathbf{s}^9$	$\mathbf{s}^{10}$
s_0^k	0	0	0	0	0	0	0	0	0	0
s_1^k	0	0	0	0	0	0	0	0	0	0
s_2^k	1	1	0	1	5	0	7	4	2	7
s_3^k	3	3	4	4	3	3	3	3	5	5
s_4^k	0	0	4	0	0	7	0	0	0	9
s_5^k	6	6	7	7	7	7	7	7	9	14
s_6^k	6	6	7	7	7	12	5	7	9	14
s_7^k	7	8	7	8	12	12	14	12	11	15
s_8^k	11	13	13	12	15	15	17	15	15	20
s_9^k	13	15	15	15	17	18	19	18	18	23

Table 3.2: A copy of Table 2.2.

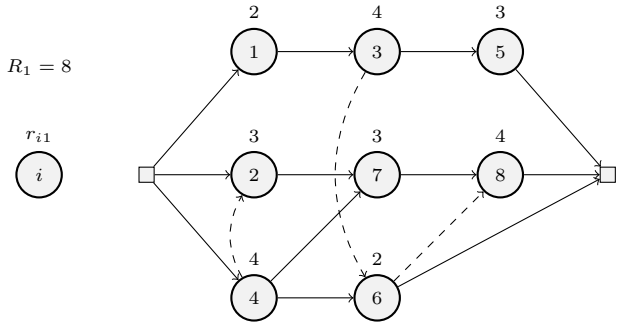


Figure 3.2: The precedence network for the example project including extra (dashed) arcs for X_1.

It is worth mentioning that, if a selection X is *sufficient*, then the induced early-start schedule for every given $\mathbf{p} \in \mathfrak{P}$, which is denoted by $ES(X, \mathbf{p})$, is feasible. However, we argue that the reverse relation is not necessarily true. In other words, the fact that $ES(X, \mathbf{p})$ is feasible for each $\mathbf{p} \in \mathfrak{P}$ is not adequate to conclude that X is sufficient. In the following, we provide a counterexample that supports the mentioned argument.

Example. *Consider the deterministic version of the instance given in the previous example. For this instance,* $\mathfrak{P} = \{\hat{\mathbf{p}}\}$ *(remember from Table 3.1 that* $\hat{\mathbf{p}} = (0, 2, 7, 3, 4, 8, 6, 4, 2, 0)$*). Even though the selection* $X_2 =$

$\{(4,2),(3,6)\}$ *is not sufficient because* $\mathscr{F}(E \cup X_2) = \{\{5,6,8\}\}$*, the schedule*

$$ES(X_2, \hat{\mathbf{p}}) = (0,0,4,2,0,5,5,11,13)$$

is feasible.

In the following, we discuss whether or not an efficient algorithm exists that determines the sufficiency of a selection. The problem of determining whether or not X is sufficient is equivalent to the problem of determining whether or not there is at least one (minimal) forbidden set for the associated instance of the RCPSP with $G(N, E \cup X)$. Both problems are solved by finding the maximum weighted anti-chain in $G(N, E \cup X)$ where weights are the resource requirements. Finding the maximum weighted anti-chain in $G(N, E \cup X)$ is equivalent to finding the maximum weight independent set (or its associated maximum weight clique) in $G(N, E \cup X)$. From Grötschel et al. (1984, Theorem 6.5) we know that there exists a polynomial-time algorithm for any maximum weight clique problem with a perfect graph. Since any partially ordered set including $G(N, E \cup X)$ is a comparability graph (which is a well-known perfect graph), we conclude that there must exist a polynomial algorithm that, given an instance of RCPSP with $G(N, E)$ and a selection X, determines whether or not X is sufficient.

A number of polynomial algorithms exist to determine the maximum weighted anti-chain (stable set) in $G(N, E \cup X)$, among which we cite the max-flow-based algorithms described in Leus (2003), Neumann et al. (2003) and Schwindt (2005). Other similar interesting results and algorithms can be found in Cong (1993), Golumbic (1980) and Kaerkes and Leipholz (1977).

3.1.2 Selection-based reactions

Since the activity durations are stochastic, robust schedules usually include buffers. Unlike many buffer insertion techniques in the literature in which buffers are inserted before the start of the activities (*e.g.*, Lambrechts et al., 2008a, 2011; Mehta and Uzsoy, 1998; Van de Vonder et al., 2007b, 2008)), we assume that the buffers are inserted after the completion times of the activities. We also assume that buffers require the same resources as their associated activities. There are two motivations behind these assumptions. First, with these two assumptions, we ensure that every vector of durations $\mathbf{p} \in \mathfrak{P}$ associates a vector of buffers $\mathbf{b}$ where $\mathbf{p} = \mathbf{p}^{\min} + \mathbf{b}$. Second, in contrast with the alternative approach, in

which buffers are inserted before the start of the activities, this approach produces no 'useless' buffers. When we insert buffers before the start of activity i, only the activities that finished immediately before activity i can use such buffers. If the resource requirements of these activities are different from the resource requirements of activity i, there are chances that these activities cannot use the inserted buffers, in which case the inserted buffers are considered 'useless'. Alternatively, when we insert buffers after the completion time (according to its minimum duration) of activity i, they become beneficial at least for activity i and are never considered 'useless'.

Given any schedule $\mathbf{s}$ and any vector of buffers $\mathbf{b}$, a unique associated sp-order can be constructed as follows. Let $\mathcal{X}_{\mathbf{s},\mathbf{b}}$ represent the sp-order induced by the pair $(\mathbf{s}, \mathbf{b})$. We define $\mathcal{X}_{\mathbf{s},\mathbf{b}}$ by $(i,j) \in \mathcal{X}_{\mathbf{s},\mathbf{b}} \Leftrightarrow s_i + p_i^{\min} + b_i \leq s_j$, where $i, j \in N$ with $i \neq j$.

A sufficient selection X is *feasible* for schedule $\mathbf{s}$ if for some realization $\mathbf{p} \in \mathfrak{P}$ (note that $\mathbf{p} = \mathbf{p}^{\min} + \mathbf{b}$), we have $X \subseteq \mathcal{X}_{\mathbf{s},\mathbf{b}} \setminus T(E)$. Here the sp-order induced by pair $(\mathbf{s}, \mathbf{0})$ becomes interesting because for every vector of buffers $\mathbf{b}$ so that schedule $\mathbf{s}$ is feasible for $\mathbf{p} = \mathbf{p}^{\min} + \mathbf{b}$, the following relation holds: $\mathcal{X}_{\mathbf{s},\mathbf{b}} \subseteq \mathcal{X}_{\mathbf{s},\mathbf{0}}$. Thus, we infer that a sufficient selection X is feasible for schedule $\mathbf{s}$ if $X \subseteq \mathcal{X}_{\mathbf{s},\mathbf{0}} \setminus T(E)$.

Definition 3.2 (Selection-based reaction)**.** *A reaction from* $\mathbf{s}$ *to* $\mathbf{s}'$ *is selection-based if there is a sufficient selection* X *that is feasible for both* $\mathbf{s}$ *and* $\mathbf{s}'$*.*

The following theorem provides the only necessary and also sufficient condition for a given reaction to be selection-based.

Theorem 3.1. *A reaction from* $\mathbf{s}$ *to* $\mathbf{s}'$ *is selection-based if and only if* $X = (\mathcal{X}_{\mathbf{s},\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}',\mathbf{0}}) \setminus T(E)$ *is sufficient.*

Proof. On one hand, we argue that $X = (\mathcal{X}_{\mathbf{s},\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}',\mathbf{0}}) \setminus T(E)$ is feasible for both $\mathbf{s}$ and $\mathbf{s}'$ because $X \subseteq \mathcal{X}_{\mathbf{s},\mathbf{0}} \setminus T(E)$ and $X \subseteq \mathcal{X}_{\mathbf{s}',\mathbf{0}} \setminus T(E)$. If X is sufficient, then the conditions of Definition 3.2 are met and the reaction from $\mathbf{s}$ to $\mathbf{s}'$ is selection-based.

On the other hand, if the reaction from $\mathbf{s}$ to $\mathbf{s}'$ is selection-based, then based on the definition there exists a selection X' that is feasible for both $\mathbf{s}$ and $\mathbf{s}'$. The immediate conclusion is that the two relations $X' \subseteq \mathcal{X}_{\mathbf{s},\mathbf{0}} \setminus T(E)$ and $X' \subseteq \mathcal{X}_{\mathbf{s}',\mathbf{0}} \setminus T(E)$ must hold and therefore

$$X' \subseteq X = (\mathcal{X}_{\mathbf{s},\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}',\mathbf{0}}) \setminus T(E)$$

also holds. It is straightforward to see that since X' is sufficient, X is must also be sufficient which concludes the proof. □

Based on Theorem 3.1, and as checking the sufficiency of a given selection can be done in polynomial time, determining whether or not a given reaction is selection-based can also be done in polynomial time.

Example. *Let us slightly change the instance given in the example of Section 3.1.2 and construct a slightly different instance. The new instance is exactly the same as the original instance except the distribution of the duration of activity* 3 *which is given as follows:*

$$\pi(\tilde{p}_3 = 2) = 0.2, \pi(\tilde{p}_3 = 3) = 0.4 \text{ and } \pi(\tilde{p}_3 = 4) = 0.4.$$

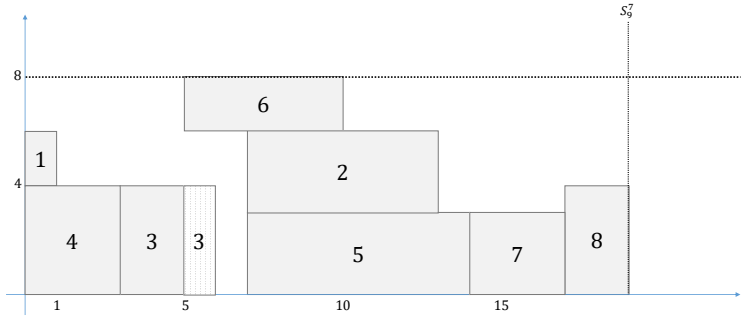
Therefore, unlike the original instance where $p_3^{\min} = 3$*, in the new instance* $p_3^{\min} = 2$.

Consider the reaction from $\mathbf{s}^7$ *to* $\mathbf{s}^9$ *for the new instance (the Gantt charts associated with these two schedules are provided in Figure 3.3). Figure 3.4(a) represents the associated graph for the transitive reduction of* $\mathcal{X}_{\mathbf{s}^7,\mathbf{0}}$ *and Figure 3.4(b) represents the associated graph for the transitive reduction of* $\mathcal{X}_{\mathbf{s}^9,\mathbf{0}}$*. Figure 3.4(c) depicts the associated graph for the transitive reduction of* $(\mathcal{X}_{\mathbf{s}^7,\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}^9,\mathbf{0}})$*. As it can be noticed in the graph of Figure 3.4(c),* $X = \mathcal{X}_{\mathbf{s}^7,\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}^9,\mathbf{0}} \setminus T(E)$ *is sufficient (i.e.,* $\mathscr{F}(E \cup X) = \emptyset$*). Since, by definition,* X *is feasible for both* $\mathbf{s}^7$ *and* $\mathbf{s}^9$*, the reaction from* $\mathbf{s}^7$ *to* $\mathbf{s}^9$ *is a selection-based reaction.*

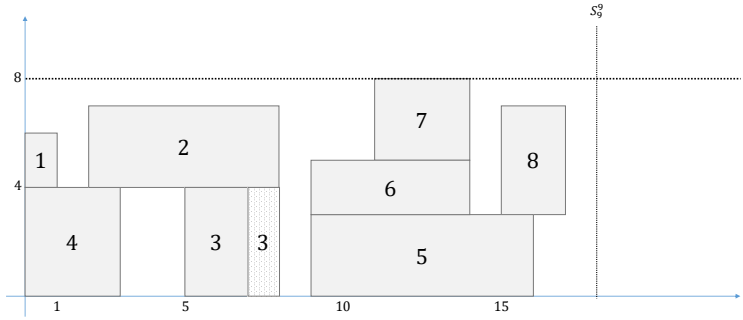
Let us consider the reaction from $\mathbf{s}^7$ *to* $\mathbf{s}^9$ *for the original instance. In this case,* $p_3^{\min} = 3$ *and the arc* $(3, 6)$ *in both graphs of Figures 3.4(a) and 3.4(c) is replaced with* $(4, 6)$*. Therefore,* X *is no longer sufficient (there exists a forbidden set:* $\{2, 3, 6\}$*) and the reaction is not selection-based.*

3.1.3 Buffer-based reactions

Despite the fact that for every selection-based reaction from $\mathbf{s}$ to $\mathbf{s}'$ there is a sufficient selection $X \subseteq (\mathcal{X}_{\mathbf{s},\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}',\mathbf{0}}) \setminus T(E)$ that is feasible for both $\mathbf{s}$ and $\mathbf{s}'$, there is no guarantee that one can actually construct schedules $\mathbf{s}$ and $\mathbf{s}'$ using selection X (we assume that selection X can construct $\mathbf{s}$ if and only if there exists a vector $\mathbf{b}$ such that $\mathbf{s} = ES(X, \mathbf{p}^{\min} + \mathbf{b})$). In this subsection, we introduce *buffer-based reactions* for which there exists a selection that can be used to construct both participating schedules.

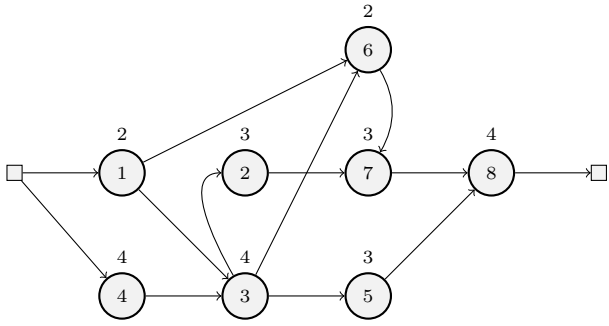


(a) Gantt chart for schedule $\mathbf{s}^7$

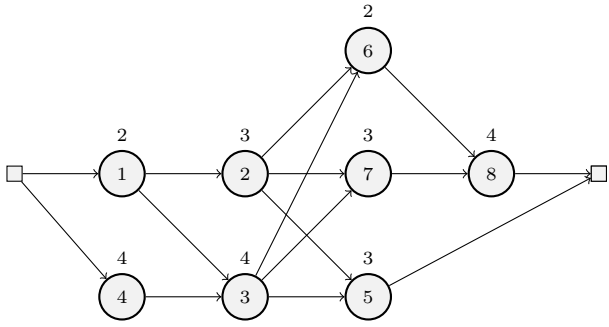


(b) Gantt chart for schedule $\mathbf{s}^9$

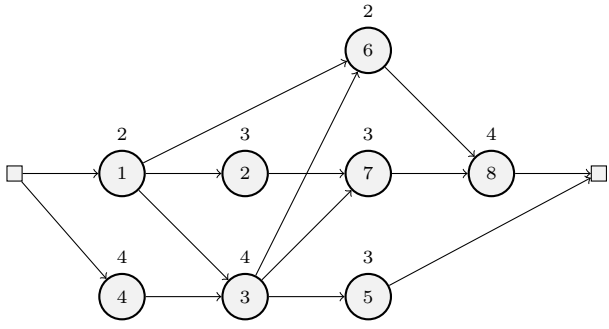
Figure 3.3: Gantt charts associated with schedules $\mathbf{s}^7$ and $\mathbf{s}^9$ in the example of Section 3.1.2. In order to produce these Gantt charts, the vector of minimum activity durations $\mathbf{p}^{\min}$ is used. Note that the minimum duration of activity 3 equals 2 for the new instance introduced for the example of Section 3.1.2 and equals 3 for the original instance.



(a) This figure represents $G(N, \bar{T}(\mathcal{X}_{\mathbf{s}^7,\mathbf{0}}))$



(b) This figure represents $G(N, \bar{T}(\mathcal{X}_{\mathbf{s}^9,\mathbf{0}}))$



(c) This figure represents $G(N, \bar{T}(\mathcal{X}_{\mathbf{s}^7,\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}^9,\mathbf{0}})) = G(N, \bar{T}(E \cup X)$

Figure 3.4: The graphs associated with the example in Section 3.1.2.

A pair $(X, \mathbf{b})$ *induces* schedule $\mathbf{s}$ if X is sufficient and $\mathbf{s} = ES(X, \mathbf{p}^{\min} + \mathbf{b})$. Similarly, a 3-tuple $(X, \mathbf{b}, \mathbf{b}')$ induces a pair of schedules $(\mathbf{s}, \mathbf{s}')$ if X is sufficient, $\mathbf{s} = ES(X, \mathbf{p}^{\min} + \mathbf{b})$ and $\mathbf{s}' = ES(X, \mathbf{p}^{\min} + \mathbf{b}')$. By definition, if pair $(X, \mathbf{b})$ induces $\mathbf{s}$, then X is sufficient and $\mathbf{s} \in \mathcal{S}(\mathbf{p}^{\min} + \mathbf{b})$ (remember from the previous chapter that $\mathcal{S}(\mathbf{p})$ represents the set of all feasible solutions for realization $\mathbf{p}$). A similar argument is true for every 3-tuple $(X, \mathbf{b}, \mathbf{b}')$ and its induced pair $(\mathbf{s}, \mathbf{s}')$. Notice that while for each 3-tuple $(X, \mathbf{b}, \mathbf{b}')$ there is exactly one induced pair of schedules, the reverse relation is rarely true.

Definition 3.3 (Buffer-based reaction)**.** *A reaction from* $\mathbf{s}$ *to* $\mathbf{s}'$ *is a buffer-based reaction if there exists a 3-tuple* $(X, \mathbf{b}, \mathbf{b}')$ *that induces* $(\mathbf{s}, \mathbf{s}')$*.*

In other words, a buffer-based reaction is a reaction in which a new schedule is created only by changing the vector of buffers (*i.e.*, the associated selection remains the same).

Example. *The selection-based reaction from* $\mathbf{s}^7$ *to* $\mathbf{s}^9$ *that has been provided in Section 3.1.2 is not a buffer-based reaction (see Section 3.1.4 in which we provide the reason why this reaction is not buffer-based). In contrast to this reaction, a reaction from* $\mathbf{s}^8$ *to* $\mathbf{s}^5$ *(see Figures 2.2(c) and 2.2(d)) is buffer-based because there exists a 3-tuple* $(X, \mathbf{b}, \mathbf{b}')$ *with*

$$\begin{aligned} X &= \{(4,2),(3,6),(6,8)\}, \\ \mathbf{b} &= (0,2,2,1,1,1,1,0,1,0) \text{ and} \\ \mathbf{b}' &= (0,2,1,1,2,1,1,0,0,0) \end{aligned}$$

that induces the pair $(\mathbf{s}^8, \mathbf{s}^5)$*.*

Let us define the class of *stochastically semi-active* schedules. This class consists of schedules, each of which is semi-active for at least one vector of durations. We label these schedules differently than the semi-active schedules since the original concept of semi-active schedules has been defined for deterministic scheduling problems.

Theorem 3.2. *For each schedule* $\mathbf{s}$*, there exists at least one pair* $(X, \mathbf{b})$ *for which* $\mathbf{s} = ES(X, \mathbf{b})$ *if and only if* $\mathbf{s}$ *is stochastically semi-active.*

Proof. On one hand, if there exists a pair $(X, \mathbf{b})$ for which $\mathbf{s} = ES(X, \mathbf{b})$, then by definition $\mathbf{s}$ is stochastically semi-active. On the other hand, if $\mathbf{s}$ is stochastically semi-active, then there is at least a vector of durations $\mathbf{p} \in \mathfrak{P}$ for which $\mathbf{s}$ is semi-active. Let $\mathbf{b} = \mathbf{p} - \mathbf{p}^{\min}$ and $X = \mathcal{X}_{\mathbf{s},\mathbf{b}} \setminus T(E)$. It is straightforward to see that X is sufficient and $\mathbf{s} = ES(X, \mathbf{b})$. □

Based on the definition of buffer-based reactions and also Theorem 3.2, if either $\mathbf{s}$ or $\mathbf{s}'$ is not stochastically semi-active, the associated reaction cannot be buffer-based. Our (initial) pool generation scheme, however, only generates stochastically semi-active schedules.

Theorem 3.3. *The class of buffer-based reactions is a subset of the class of selection-based reactions.*

Proof. If a reaction from $\mathbf{s}$ to $\mathbf{s}'$ is buffer-based, then by definition there is a selection X that is sufficient and that is feasible for both schedules $\mathbf{s}$ and $\mathbf{s}'$. Therefore this reaction is also selection-based. □

From Theorem 3.3, we also infer that any reaction that is not selection-based is certainly not buffer-based. However, in general, determining whether or not a reaction from $\mathbf{s}$ to $\mathbf{s}'$ is buffer-based is not trivial. We need to either find one associated 3-tuple $(X, \mathbf{b}, \mathbf{b}')$ that induces $(\mathbf{s}, \mathbf{s}')$ or to prove that there exists no such 3-tuple. We introduce the following decision problem:

Problem BBP
Instance: A PR-RCPSP instance $(N, E, \tilde{\mathbf{p}}, \mathbf{r}, \mathcal{R})$ and a pair $(\mathbf{s}, \mathbf{s}')$.
Question: Does there exist a 3-tuple $(X, \mathbf{b}, \mathbf{b}')$ that induces $(\mathbf{s}, \mathbf{s}')$?

If the answer to BBP is 'YES', then the reaction from $\mathbf{s}$ to $\mathbf{s}'$ is buffer-based. Otherwise, if the answer is 'NO', the reaction is not buffer-based. Currently, the complexity of this problem is still open. Nevertheless, in Section 3.1.4, we provide an exponential time algorithm that solves BBP.

3.1.4 An implicit enumeration algorithm

As stated before, if the reaction is not selection-based, then the answer to problem BBP is a definite 'NO'. If however the reaction is selection-based we must enumerate all 3-tuples $(X, \mathbf{b}, \mathbf{b}')$ with $X \subseteq (\mathcal{X}_{\mathbf{s},\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}',\mathbf{0}}) \setminus T(E)$ and see if at least one of these 3-tuples satisfies the condition of Definition 3.3. In the following, we provide an implicit enumeration algorithm.

Consider schedule $\mathbf{s}$ and pair $(X, \mathbf{b})$. Activity j *pushes* activity i in schedule $\mathbf{s}$ for pair $(X, \mathbf{b})$ if $s_j + p_j^{min} + b_j = s_i$ and $(j, i) \in T(X \cup E)$. Let $\mathcal{A}(\mathbf{s}, X, \mathbf{b})$ be the set of all *pushing pairs* (j, i) where j pushes i in schedule $\mathbf{s}$ for pair $(X, \mathbf{b})$. The set $\mathcal{A}(\mathbf{s}, X, \mathbf{b})$ is called a *complete pushing set* if for each $i \in N \setminus \{0\}$ there is at least one $(j, i) \in \mathcal{A}(\mathbf{s}, X, \mathbf{b})$. The following theorem holds.

Theorem 3.4. $\mathbf{s} = ES(X, \mathbf{p}^{\min}+\mathbf{b})$ *if and only if* $\mathcal{A}(\mathbf{s}, X, \mathbf{b})$ *is a complete pushing set.*

Proof. If $\mathbf{s} = ES(X, \mathbf{p}^{\min}+\mathbf{b})$, then by definition each activity $i \in N \setminus \{0\}$ starts at the completion time of another activity $j \in N \setminus \{n+1\}$ and thus $\mathcal{A}(\mathbf{s}, X, \mathbf{b})$ is a complete pushing set. On the other hand, if $\mathcal{A}(\mathbf{s}, X, \mathbf{b})$ is a complete pushing set, then $s_j + p_j^{min} + b_j = s_i$ for each $(j, i) \in \mathcal{A}(\mathbf{s}, X, \mathbf{b})$. We also know that $s_j + p_j^{min} + b_j \leq s_i$ for each $(j, i) \in T(X \cup E)$. Therefore, we conclude

$$s_i = \max_{(i,j) \in T(X \cup E)} \{s_j + p_j^{min} + b_j\}$$

which suffices to infer that $\mathbf{s} = ES(X, \mathbf{p}^{\min} + \mathbf{b})$. Note that we assume that the dummy start activity always starts and finishes at time zero. □

Based on Theorem 3.4, we deduce that if both complete pushing sets $\mathcal{A}(\mathbf{s}, X, \mathbf{b})$ and $\mathcal{A}(\mathbf{s}', X, \mathbf{b}')$ exist, then a 3-tuple $(X, \mathbf{b}, \mathbf{b}')$ that induces $(\mathbf{s}, \mathbf{s}')$ also exists.

We define a set B of activities ($B \subseteq N \setminus \{0, n+1\}$) that represents all activities that are evidently pushed by the dummy start activity (*i.e.*, all activities that start at time zero in both schedules). We construct an *activity list* (AL) by sorting all activities in $N \setminus (B \cup \{0\})$ based on the non-decreasing order of their starting times in one of the schedules (for instance the schedule from which we react). As a breaking rule, the activity with the smaller index is placed before the activity with the larger index. Note that AL is an array with $n - |B| + 1$ members and AL_k refers to the activity in the kth position in AL.

We propose a branching tree in which each node (which is denoted by $\mathcal{N}_u$ where u is the index of the node, indicating the sequence in which the nodes are visited) represents a pair of partially constructed pushing sets $(\mathcal{A}_u, \mathcal{A}'_u)$ with $\mathcal{A}_u$ being the associated set of pushing pairs for $\mathbf{s}$ and $\mathcal{A}'_u$ being the associated set of pushing pairs for $\mathbf{s}'$. The first two levels are associated with the first activity in the AL, the second two levels are associated with the second activity in the AL and so on. Finally the last two levels are associated with the last activity in the AL. Therefore, this tree consists of $2n$ levels. In each odd level of the tree, we add one pushing pair (j, i) to $\mathcal{A}_u$ and in each even level, we add one pushing pair (j', i') to $\mathcal{A}'_u$ where i and i' are the associated activities for the two levels, respectively.

Given any pair of partially constructed pushing sets $(\mathcal{A}_u, \mathcal{A}'_u)$, we construct an associated 3-tuple $(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{b}(\mathcal{A}_u), \mathbf{b}'(\mathcal{A}_u))$ with $X(\mathcal{A}_u, \mathcal{A}'_u)$

being the associated selection and $\mathbf{b}(\mathcal{A}_u)$ and $\mathbf{b}'(\mathcal{A}_u)$ being the associated vectors of buffers. In the following, we explain how we compute $X(\mathcal{A}_u, \mathcal{A}'_u)$, $\mathbf{b}(\mathcal{A}_u)$ and $\mathbf{b}'(\mathcal{A}_u)$. Let $X^a = (\mathcal{X}_{\mathbf{s},\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}',\mathbf{0}}) \setminus T(E)$ be the set of all arcs that are candidates to be included in any selection associated to $(\mathcal{A}_u, \mathcal{A}'_u)$. By assuming that j pushes i in $\mathbf{s}$ (which is equivalent to adding (j,i) to $\mathcal{A}_u$), we also accept that all arcs (j,k) in $T(E \cup X^a)$ such that $s_i > s_k$ (note that $s_j + p_j^{\min} + b_j(\mathcal{A}_u) = s_i$) are violated in $\mathbf{s}$. The set of all violating arcs that are induced by all pushing pairs in $\mathcal{A}_u$ is referred to as $V(\mathcal{A}_u)$. We obtain $V(\mathcal{A}'_u)$ in a similar way. The associated selection $X(\mathcal{A}_u, \mathcal{A}'_u)$ is computed as follows:

$$X(\mathcal{A}_u, \mathcal{A}'_u) = X^a \setminus (V(\mathcal{A}_u) \cup V(\mathcal{A}'_u)).$$

Moreover, for each partially constructed pushing set $\mathcal{A}_u$, we compute its associated set of buffers which is denoted by $\mathbf{b}(\mathcal{A}_u)$ as follows:

$$b_j(\mathcal{A}_u) = \begin{cases} 0 & \text{if } \forall i \in N \setminus \{0,j\}, (j,i) \notin \mathcal{A}_u \\ s_i - s_j - p_j^{\min} & \text{if } \exists i \in N \setminus \{0,j\}, (j,i) \in \mathcal{A}_u \end{cases}.$$

With a similar reasoning, we can obtain $\mathbf{b}'(\mathcal{A}_u)$.

Not every possible choice of $(\mathcal{A}_u, \mathcal{A}'_u)$ is *feasible*. A pair of pushing sets $(\mathcal{A}_u, \mathcal{A}'_u)$ is feasible if its associated 3-tuple $(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{b}(\mathcal{A}_u), \mathbf{b}'(\mathcal{A}_u))$ induces $(\mathbf{s}, \mathbf{s}')$. In the following, we propose five conditions that must be met to conclude that $(\mathcal{A}_u, \mathcal{A}'_u)$ is feasible. The proofs for the necessity of these five conditions are given in Theorem 3.5.

Condition 3.1. *For every $i, i', j \in N$ such that $(j,i) \in \mathcal{A}_u$ and $(j,i') \in \mathcal{A}_u$, the equality $s_i = s_{i'}$ holds. Also, for every $i, i', j \in N$ such that $(j,i) \in \mathcal{A}'_u$ and $(j,i') \in \mathcal{A}'_u$, the equality $s'_i = s'_{i'}$ holds.*

Condition 3.2. $\mathbf{s}$ *and* $\mathbf{s}'$ *are resource feasible for* $\mathbf{p}^{\min} + \mathbf{b}(\mathcal{A}_u)$ *and* $\mathbf{p}^{\min} + \mathbf{b}'(\mathcal{A}'_u)$*, respectively.*

Condition 3.3. $(V(\mathcal{A}_u) \cup V(\mathcal{A}'_u)) \cap T(E) = \emptyset$.

Condition 3.4. $X(\mathcal{A}_u, \mathcal{A}'_u)$ *is sufficient.*

Condition 3.5. *Both $\mathcal{A}_u$ and $\mathcal{A}'_u$ are complete pushing sets.*

The following theorem holds.

Theorem 3.5. $(\mathcal{A}_u, \mathcal{A}'_u)$ *is feasible if and only if Conditions 3.1 to 3.5 are met.*

Proof. This proof consists of two parts. In the first part, we prove the necessity of these five conditions whereas in the second part we prove the sufficiency of the combination of these five conditions. Note that as we mentioned earlier, a pair $(\mathcal{A}_u, \mathcal{A}'_u)$ is feasible if its associated 3-tuple $(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{b}(\mathcal{A}_u), \mathbf{b}'(\mathcal{A}_u))$ induces $(\mathbf{s}, \mathbf{s}')$.

In the following, we provide reasons why the violation of each of Conditions 3.1 to 3.5 guarantees the infeasibility of $(\mathcal{A}_u, \mathcal{A}'_u)$.

- If $i, i', j \in N$ such that $(j, i) \in \mathcal{A}_u$ and $(j, i') \in \mathcal{A}_u$, then whatever value $b_j(\mathcal{A}_u)$ takes we have

$$s_j + p_j^{\min} + b_j(\mathcal{A}_u) = s_i = s_{i'}.$$

  A similar argument is true if $i, i', j \in N$ such that $(j, i) \in \mathcal{A}'_u$ and $(j, i') \in \mathcal{A}'_u$, for which case s'_i must be equal to $s'_{i'}$. Notice that any pair $(\mathcal{A}_u, \mathcal{A}'_u)$ that violates Condition 3.1 contradicts the above arguments.

- If Condition 3.2 does not hold, then any 3-tuple $(X, \mathbf{b}, \mathbf{b}')$ such that $\mathcal{A}_u \subseteq \mathcal{A}(\mathbf{s}, X, \mathbf{b})$ and $\mathcal{A}'_u \subseteq \mathcal{A}(\mathbf{s}', X, \mathbf{b}')$ cannot induce $(\mathbf{s}, \mathbf{s}')$ simply because the pushing pairs in $\mathcal{A}_u$ and $\mathcal{A}'_u$ (which also exist in $\mathcal{A}(\mathbf{s}, X, \mathbf{b})$ and $\mathcal{A}(\mathbf{s}', X, \mathbf{b}')$, respectively) cause resource infeasibilities in schedule $\mathbf{s}$ and/or schedule $\mathbf{s}'$.

- If Condition 3.3 does not hold, then the pushing pairs in $\mathcal{A}_u$ and $\mathcal{A}'_u$ cause the violation of some of the precedence constraints in $T(E)$ and thus any 3-tuple $(X, \mathbf{b}, \mathbf{b}')$ such that $\mathcal{A}_u \subseteq \mathcal{A}(\mathbf{s}, X, \mathbf{b})$ and $\mathcal{A}'_u \subseteq \mathcal{A}(\mathbf{s}', X, \mathbf{b}')$ cannot induce $(\mathbf{s}, \mathbf{s}')$.

- Condition 3.4 must not be violated because otherwise no selection $X \subseteq X(\mathcal{A}_u, \mathcal{A}'_u)$ is sufficient and thus $(\mathbf{s}, \mathbf{s}')$ cannot be induced.

- Both $\mathcal{A}_u$ and $\mathcal{A}'_u$ must be complete pushing sets because otherwise based on Theorem 3.4 we have $\mathbf{s} \neq ES(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{p}^{\min} + \mathbf{b}(\mathcal{A}_u))$ and/or $\mathbf{s}' \neq ES(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{p}^{\min} + \mathbf{b}'(\mathcal{A}_u))$.

In the second part of the proof, we provide adequate reasonings that support the sufficiency of the combination of the above five conditions to conclude the feasibility of $(\mathcal{A}_u, \mathcal{A}'_u)$. When Conditions 3.1 to 3.4 hold, we can infer that both schedules resulted from $ES(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{p}^{\min} + \mathbf{b}(\mathcal{A}_u))$ and $ES(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{p}^{\min} + \mathbf{b}'(\mathcal{A}_u))$ are stochastically semi-active (because of Condition 3.1), resource feasible (because of Conditions 3.2 and 3.4)

and precedence feasible (because of Condition 3.3). When Condition 3.5 holds, based on Theorem 3.4 we have

$$\mathbf{s} = ES(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{p}^{\min} + \mathbf{b}(\mathcal{A}_u)) \text{ and}$$
$$\mathbf{s}' = ES(X(\mathcal{A}_u, \mathcal{A}'_u), \mathbf{p}^{\min} + \mathbf{b}'(\mathcal{A}_u)).$$

Therefore, we conclude that Conditions 3.1 to 3.5 are sufficient to determine the feasibility of $(\mathcal{A}_u, \mathcal{A}'_u)$. □

Each node $(\mathcal{A}_u, \mathcal{A}'_u)$ in an even level l (including level zero ($l = 0$)) is branched into a number of nodes $(\mathcal{A}_u \cup \{(j,i)\}, \mathcal{A}'_u)$ in level $l+1$ (which is an odd level) with associated activity $i = \mathrm{AL}_{\lceil (l+1)/2 \rceil}$ where $(j,i) \in T(E \cup X(\mathcal{A}_u, \mathcal{A}'_u))$. Likewise, each node $(\mathcal{A}_u, \mathcal{A}'_u)$ in an odd level l' is branched into a number of nodes $(\mathcal{A}_u, \mathcal{A}'_u \cup \{(j',i')\})$ in level $l'+1$ (which is an even level) with an associated activity $i' = \mathrm{AL}_{\lceil (l'+1)/2 \rceil}$ where $(j',i') \in T(E \cup X(\mathcal{A}_u, \mathcal{A}'_u))$.

Theorem 3.6. *The violation of each one of Conditions 3.1 to 3.4 in node $(\mathcal{A}_u, \mathcal{A}'_u)$ guarantees that no direct or transitive child of $(\mathcal{A}_u, \mathcal{A}'_u)$ is feasible. Therefore, such a node must be fathomed.*

Proof. The proof becomes apparent by referring to the first four bullets in the proof of Theorem 3.5. □

The branching starts with the root node $\mathcal{N}_0 : (\mathcal{A}_0, \mathcal{A}'_0) = (\emptyset, \emptyset)$. The root node is branched into a number of nodes $(\{(j,i)\}, \emptyset)$, each associated with one possible pushing pair (j,i) to be added to $\mathcal{A}_u$. In the second level, each node $(\{(j,i)\}, \emptyset)$ is branched into a number of nodes $(\{(j,i)\}, \{(j',i)\})$, each associated with one possible pushing pair (j',i) to be added to $\mathcal{A}'_u$. In the other levels we continue branching in the same fashion as in the first two levels. Backtracking occurs if at least one of the Conditions 3.1 to 3.4 is violated (see Theorem 3.6). As soon as a pair of complete pushing sets is constructed (which happens when we visit a feasible node on the level $2(n - |B| + 1)$), the algorithm immediately halts and the answer to problem BBP is 'YES'. If after visiting all nodes in the tree no such pair of complete pushing sets is constructed, the answer to problem BBP is 'NO'.

Example. *Consider the selection-based reaction from $\mathbf{s}^7$ and $\mathbf{s}^9$ for the slightly changed instance provided in Section 3.1.2. In order to determine whether this reaction is also buffer-based or not, we use the algorithm described above to implicitly enumerate all potential 3-tuples $(X, \mathbf{b}^7, \mathbf{b}^9)$.*

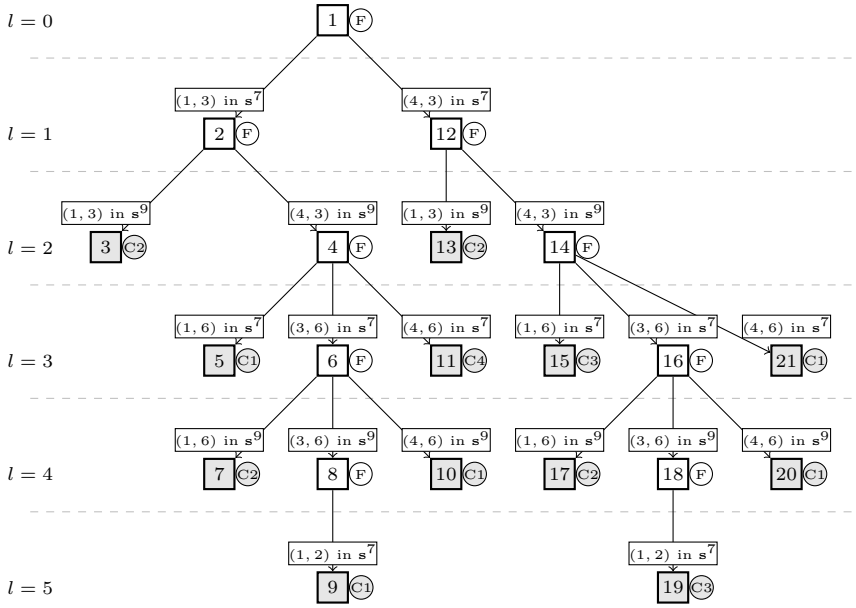


Figure 3.5: The tree associated with the example in Section 3.1.4.

We construct AL by ordering activities based on their starting times in one of the schedules (in this case schedule $\mathbf{s}^7$*) and removing all activities that start at time zero (these activities are pushed by the dummy start activity):*

$$\mathrm{AL} = (3, 6, 2, 5, 7, 8, 9).$$

We also compute $X = (\mathcal{X}_{\mathbf{s}^7,\mathbf{0}} \cap \mathcal{X}_{\mathbf{s}^9,\mathbf{0}}) \setminus T(E)$ *as follows:*

$$X = \{(1,2), (1,6), (1,7), (1,8), (3,6), (3,7), (3,8), (4,3), (4,5)\}.$$

Figure 3.5 illustrates the tree associated with this example. The label above each node indicates the pushing pair that is added to the partially constructed pushing sets. The number inside each node represents the index of the node (notice that nodes are visited in the non-decreasing order of their indices). The associated activity in level one and level two is $\mathrm{AL}_{\lceil 1/2 \rceil} = \mathrm{AL}_{\lceil 2/2 \rceil} = 3$*, the associated activity in level three and level four is* $\mathrm{AL}_{\lceil 3/2 \rceil} = \mathrm{AL}_{\lceil 4/2 \rceil} = 6$ *and the associated activity in level five is*

$\text{AL}_{\lceil 5/2 \rceil} = 2$. *The branching starts with the root node* $\mathcal{N}_1 : (\mathcal{A}_1^7, \mathcal{A}_1^9) = (\emptyset, \emptyset)$ *in level zero. The root node is branching into two nodes: the first child node* $\mathcal{N}_2 : (\mathcal{A}_2^7, \mathcal{A}_2^9) = (\{(1,3)\}, \emptyset)$ *enforces activity* 3 *to be pushed by activity* 1 *in* $\mathbf{s}^7$ *and the second child node* $\mathcal{N}_{12} : (\{(4,3)\}, \emptyset)$ *enforces activity* 3 *to be pushed by activity* 4 *in* $\mathbf{s}^7$. *Likewise,* $\mathcal{N}_2$ *is branched into two nodes: the first child node* $\mathcal{N}_3 : (\{(1,3)\}, \{(1,3)\})$ *enforces activity* 3 *to be pushed by activity* 1 *in both* $\mathbf{s}^7$ *and* $\mathbf{s}^9$ *whereas the second child node* $\mathcal{N}_4 : (\{(1,3)\}, \{(4,3)\})$ *enforces activity* 3 *to be pushed by activity* 1 *in* $\mathbf{s}^7$ *and by activity* 4 *in* $\mathbf{s}^9$. *All other nodes are branched in the same fashion.*

All nodes with an 'F' letter are associated with feasible partially constructed sets of pushing pairs. All nodes with a 'C' letter are nodes that are infeasible due to one of the conditions discussed above. For instance, node $\mathcal{N}_5$ *violates Condition 3.1 because activity* 1 *pushes both activities* 3 *and* 6 *in* $\mathbf{s}^7$, *yet* $s_3^7 \neq s_6^7$. *Node* $\mathcal{N}_3$ *violates Condition 3.2 because* $\mathbf{s}^9$ *is not resource feasible for* $\mathbf{p}^{\min} + \mathbf{b}^9(\mathcal{A}_3^9)$ *at time* 2 *when activities* 1 *and* 4 *are ongoing and there are not enough available resources to start activity* 2. *Node* $\mathcal{N}_{15}$ *violates Condition 3.3 because* $(V(\mathcal{A}_{15}^7) \cup V(\mathcal{A}_{15}^9)) \cap T(E) = \{(1,3)\}$. *Finally, node* $\mathcal{N}_{11}$ *violates Condition 3.4 because*

$$\begin{aligned} X(\mathcal{A}_{11}^7, \mathcal{A}_{11}^9) =& X^a \setminus \{(4,3)\} \\ =& \{(1,2),(1,6),(1,7),(1,8),(3,6),(3,7),(3,8),(4,5)\} \end{aligned}$$

is not sufficient (there exists a forbidden set $\{2,3,4\}$*). As after visiting all nodes in the tree, no pair of complete pushing sets is constructed, we conclude that the answer to problem BBP is 'NO' and thus the reaction from* $\mathbf{s}^7$ *to* $\mathbf{s}^9$ *is not buffer-based.*

3.1.5 Computational results

In this section, we run an experiment to evaluate the importance of selection-based reactions and buffer-based reactions. In this experiment, we use Model 3 proposed in the previous chapter to obtain a PR-policy. We set $\lambda = 1, \kappa_1 = 500, \kappa_2 = 50$ and $\kappa_3 = 4$.

In Table 3.3, we report the ratio (in percentage) of the selection-based reactions to all reactions (SB), the ratio (in percentage) of selected selection-based reactions to all selected reactions (SBS), the ratio (in percentage) of selection-based reactions in the optimal PR-policy to all reactions in the optimal PR-policy (SBO) and the ratio $\frac{\text{SBO}}{\text{SB}}$ (which is a measure of importance) for different choices of w_b and w_r. Note that a reaction that is selected to be the best among its rivals, might not be part

w_b	w_r	SB	SBS	SBO	$\frac{\text{SBO}}{\text{SB}}$
25	0	48.57	94.34	98.87	2.04
	50	48.57	92.30	98.21	2.02
	100	48.57	90.00	97.67	2.01
50	0	48.57	94.34	98.69	2.03
	50	48.57	92.30	98.69	2.03
	100	48.57	90.00	98.38	2.03

Table 3.3: The ratio (in percentage) of the selection-based reactions to all reactions.

of the optimal solution. We observe that about half of the reactions in the network are selection-based. The fact that this number is so high is probably because of the way we generate the set of schedules. The results in Table 3.3 indicate that more than 90 percent of the reactions that are selected and more than 97 percent of the reactions in the optimal PR-policy are selection-based. By looking at the ratio $\frac{\text{SBO}}{\text{SB}}$, we notice that the chance of having a selection-based reaction in the optimal policy is more than two times larger than the chance of having a selection-based reaction in the whole network. This fact emphasizes even further the importance of selection-based reactions and suggests that studying these reactions and then focusing on generating them is one promising way to find higher-quality PR-policies.

To just get an idea of how much focusing on selection-based reactions will benefit us, we slightly modify Model 3 where we remove all non-selection-based reactions from the network. This model is called SB-Model 3 and its associated combined cost is referred to as SBCC. Table 3.4 reports some results on the comparison between Model 3 and SB-Model 3. As is shown in the table, by allowing only selection-based reactions, we on average lose less than 0.03 percent in quality while we need to solve a much smaller network, which is quite promising.

Within the class of selection-based reactions, the buffer-based reactions seem to be even more important. In Table 3.5, we report the ratio (in percentage) of the buffer-based reactions to all reactions (BB), the ratio (in percentage) of selected buffer-based reactions to all selected reactions (BBS), the ratio (in percentage) of buffer-based reactions in the optimal PR-policy to all reactions in the optimal PR-policy (BBO) and the ratio $\frac{\text{BBO}}{\text{BB}}$ for different choices of w_b and w_r. These results indicate that only around 35 percent of the reactions are buffer-based. Nevertheless,

w_b	w_r	SBCC	CC	DEV
25	0	1692.26	1691.87	0.03
	50	1820.07	1819.70	0.02
	100	1899.88	1899.39	0.03
50	0	3278.01	3277.43	0.02
	50	3437.57	3436.95	0.02
	100	3557.78	3557.43	0.01

Table 3.4: The average combined cost when only selection-based reactions are considered (SBCC), that when all reactions are considered (CC) and the average deviation (in percentage) of SBCC from CC.

w_b	w_r	BB	BBS	BBO	$\frac{\text{BBO}}{\text{BB}}$
25	0	34.90	73.35	88.92	2.55
	50	34.90	70.85	86.44	2.48
	100	34.90	68.90	85.35	2.45
50	0	34.90	73.35	88.55	2.54
	50	34.90	70.85	88.09	2.52
	100	34.90	68.90	86.84	2.49

Table 3.5: The ratio (in percentage) of the buffer-based reactions to all reactions.

these reactions do have a considerably large contribution in the optimal PR-policy (85 to 90 percent). By comparing the ratios $\frac{\text{BBO}}{\text{BB}}$ and $\frac{\text{SBO}}{\text{SB}}$, we may infer that buffer-based reactions are slightly more important than selection-based reactions. However, this conclusion is not completely fair since a large percentage (around 70 percent) of selection-based reactions are also buffer-based. Therefore, we introduce the class of *selection-but-not-buffer-based* reactions which contains all selection-based reactions that are not buffer-based.

Table 3.6 reports the ratio (in percentage) of the selection-but-not-buffer-based reactions to all reactions (SNB), the ratio (in percentage) of selected selection-but-not-buffer-based reactions to all selected reactions (SNBS), the ratio (in percentage) of selection-but-not-buffer-based reactions in the optimal PR-policy to all reactions in the optimal PR-policy (SNBO) and the ratio $\frac{\text{SNBO}}{\text{SNB}}$ for different choices of w_b and w_r. Notice that the class of buffer-based reactions and the class of selection-but-not-buffer-based reactions combined construct the class of selection-based reactions.

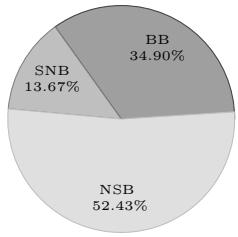
w_b	w_r	SNB	SNBS	SNBO	$\frac{\text{SNBO}}{\text{SNB}}$
25	0	13.67	21.00	9.95	0.73
	50	13.67	21.45	11.77	0.86
	100	13.67	21.10	12.32	0.90
50	0	13.67	21.00	10.14	0.74
	50	13.67	21.45	10.60	0.76
	100	13.67	21.10	11.54	0.84

Table 3.6: The ratio (in percentage) of the selection-but-not-buffer-based reactions to all reactions.

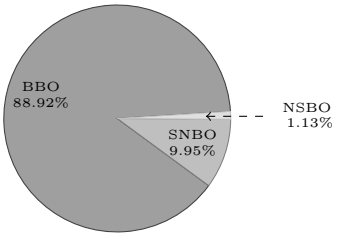
We observe that around 15 percent of all reactions are selection-but-not-buffer-based reactions. Although, these reactions are selection-based, they have a relatively small contribution (around 10 to 13 percent) in the optimal PR-policy. By looking at ratios $\frac{\text{BBO}}{\text{BB}}$ and $\frac{\text{SNBO}}{\text{SNB}}$, we observe that buffer-based reactions are considerably more important than selection-but-not-buffer-based reactions: the chance of having a buffer-based reaction in the optimal policy is almost 3 times larger than the chance of having a selection-but-not-buffer-based reaction in the optimal policy.

To further understand the importance of the buffer-based reactions, we report part of the information provided in Tables 3.3, 3.5 and 3.6 using a pie chart representation. In Figure 3.6, we depict the contributions of three mutually exclusive and collectively exhaustive classes of reactions in the network of Model 3 and in its associated optimal PR-policy for the problem setting where $w_b = 25$ and $w_r = 0$.

The classes under comparison are the class of non-selection based (NSB) reactions, the class of buffer-based (BB) reactions and the class of selection-but-not-buffer-based (SNB) reactions. Figure 3.6(a) shows the contributions of these classes of reactions in the whole network and Figure 3.6(b) displays the contributions of these classes of reactions in the associated optimal PR-policy. The futility of non-selection-based reactions in the optimal PR-policy is very clear in Figure 3.6 (NSBO represents the percentage of non-selection-based reactions in the optimal PR-policy): although 52.43 percent of reactions are non-selection-based, the contribution of these reactions in the optimal PR-policy is only 1.13 percent. It is also clear that buffer-based reactions are very important: as stated before, despite the fact that only 34.90 percent of reactions are buffer-based, their contribution in the optimal PR-policy is very high (88.92 percent).



(a) The average contributions of different classes of reaction in the whole network



(b) The average contributions of different classes of reaction in the optimal PR-policy

Figure 3.6: The average contributions of different classes of reaction for the setting where $w_b = 25$ and $w_r = 0$.

w_b	w_r	BBCC	CC	DEV
25	0	1701.98	1691.87	0.67
	50	1826.28	1819.70	0.40
	100	1904.10	1899.39	0.27
50	0	3288.60	3277.43	0.39
	50	3444.49	3436.95	0.24
	100	3563.02	3557.43	0.17

Table 3.7: The average combined cost when only buffer-based reactions are considered (BBCC), that when all reactions are considered (CC) and the average deviation (in percentage) of BBCC from CC.

To indicate how much we can rely solely on buffer-based reactions, we slightly modify Model 3 where we remove all non-buffer-based reactions from the network. This model is called BB-Model 3 and its associated combined cost is referred to as BBCC. Table 3.7 reports some results on the comparison between Model 3 and BB-Model 3. As is shown in the table, by focusing only on buffer-based reactions, we lose on average between 0.17 to 0.67 percent in quality.

The importance of buffer-based reactions varies for different classes of instances. Table 3.8 reports the ratios BB, BBS, BBO and $\frac{\text{BBO}}{\text{BB}}$ for different values of NC (network complexity), for different values of RF (resource factor) and for different values of RS (resource strength). The parameter NC reflects the average number of non-redundant precedence relations per activity including dummy activities, the parameter RF represents the

		BB	BBS	BBO	$\frac{\text{BBO}}{\text{BB}}$
NC	1.5	33.26	70.51	87.12	2.62
	1.8	32.83	68.92	85.30	2.60
	2.1	38.61	73.67	89.68	2.32
RF	0.25	50.01	77.67	90.42	1.81
	0.5	32.73	70.17	84.86	2.59
	0.75	25.51	68.78	88.28	3.46
	1	31.34	67.51	85.91	2.74
RS	0.2	19.07	80.70	94.27	4.94
	0.5	19.72	59.75	82.22	4.17
	0.7	28.91	63.22	84.87	2.94
	1	71.90	80.47	88.10	1.23

Table 3.8: The ratio (in percentage) of the buffer-based reactions to all reactions for different classes of instances.

average portion of resources used and consumed and the parameter RS measures the strength of resource constrainedness. We refer interested readers to Kolisch and Sprecher (1997) where these parameters are introduced. Comparing the ratios BB, BBS, BBO and $\frac{\text{BBO}}{\text{BB}}$, we observe that the importance of buffer-based reactions remains almost constant when choosing different values for NC. When RF is increased, the ratio BBO stays between 85 percent and 90 percent. However, the ratio BB first decreases (down to 25.51 percent) and then increases and therefore, $\frac{\text{BBO}}{\text{BB}}$ first increases (up to 3.446) and then decreases. When RS is increased, on one hand, the ratio BBO first decreases (down to 82.22 percent) and then increases, and on the other hand, the ratio BB increases rapidly (specially when RS changes between 0.5 and 1). Taking both trends into consideration, we notice that by increasing RS the ratio $\frac{\text{BBO}}{\text{BB}}$ continuously decreases. All in all, we conclude that the buffer-based reactions are of the greatest importance when RF = 0.75 and RS = 0.2.

3.1.6 Discussion

The results presented in Section 3.1.5 confirm the importance of the selection-based and buffer-based reactions, both of which put emphasis on the unique selection of resource arcs (resource flow) between schedules, in optimal PR-policies. One might wonder how and to what extent we can use this important property to construct a set of simple and com-

pact rules that suggest proper reactions for infeasible situations without actually solving the PR-RCPSP. The idea of finding a set of simple and compact rules seems very similar to that of policies introduced in Section 1.2.1 within the context of the SRCPSP.

Following the results in the previous subsection, we observe that the probability that a reaction in an optimal PR-policy is selection-based is very high. This suggests that finding an appropriate selection based on which the reactions are made is very promising. Such a selection and its application very much resembles the set of added resource arcs associated with an early start policy (remember that the class of early start policies is one of the classes of policies introduced in Section 1.2.1). Let us introduce the class of *early start PR-policies* as follows: an early start PR-policy is represented by $(X, \mathbf{s}^{base}, \mathfrak{M})$ where X is a sufficient selection that also represents the additional resource arcs to be added to the precedence network, $\mathbf{s}^{base}$ represents a baseline schedule and $\mathfrak{M}$ is a mechanism that appropriately introduces buffers (which is crucially important when reaction costs are large). Aside from the process of finding X and $\mathbf{s}^{base}$ that seems to be very difficult, developing $\mathfrak{M}$ appears to be very challenging. Therefore, what is discussed in this subsection is solely an attempt to introduce a future area of research.

Despite the fact that most of the reactions in an optimal PR-policy are selection-based, the number of unique associated selections in an optimal PR-policy can be very large and therefore searching for an early start PR-policy that is described in the previous paragraph seems to be very naive. Alternatively, we introduce another class of PR-policies. Consider a list L of $|\mathcal{F}|$ items where each item is associated with one forbidden set in $\mathcal{F}$. List L *advocates* a sufficient selection X if for each $FS \in \mathcal{F}$ there exists a pair $(i, j) \in X$ such that $i, j \in FS$ and j is the item in L that is associated with FS. If one single list advocates all members of a set of selections, then we may argue that such a set of selections resembles a pre-selective policy introduced in Section 1.2.1 (note that a pre-selective policy advocates a group of ES-policies that are slightly different from each other). Let us introduce the class of *pre-selective PR-policies* as follows: each pre-selective PR-policy is represented by $(L, \mathbf{s}^{base}, \mathfrak{M})$. We believe that the following conjecture holds:

Conjecture 3.1. *The following statements are true:*

- *An instance of SRCPSP can be converted to an instance of PR-RCPSP and can be optimally solved by any method that optimally*

solves PR-RCPSP. The optimal PR-policy associated with such an instance is the optimal policy to the SRCPSP.

- *The counterparts of all classes of policies introduced in Section 1.2.1 are subsets of the class of all PR-policies.*

3.2 The selection of schedules

To obtain a high-quality PR-policy, the choice of the input set of schedules is very important. On one hand, since one of the very important decisions is the choice of the baseline schedule, the set of schedules must contain a number of potential baseline schedules that are considerably different from each other. On the other hand, we need schedules that are closely-related (share very similar cuts and/or continuations) to increase the probability that reactions are selection-based or possibly buffer-based. These two conditions make the process of selecting the set of schedules very challenging.

3.2.1 A schedule refinement technique

We focus on generating high-quality sets of schedules for which our models, proposed in the previous chapter, output higher quality PR-policies. To achieve this goal, a multi-stage technique is proposed to gradually refine and then reconstruct the set of initial schedules such that the resulting PR-policy becomes less costly. The motivation behind choosing such a multi-stage technique lies in the fact that finding a high-quality neighborhood of schedules is very important in obtaining high-quality PR-policies. The multi-stage technique that is proposed in this section outputs a much less diverse neighborhood of high-quality initial schedules from which the final set of schedules is constructed. Therefore, in the model associated with the final set of schedules, the probability of having selection-based reactions and the probability of having buffer-based reactions are much higher than those when multi-stage technique is not used.

Our proposed multi-stage method is detailed in Algorithm 3.1. In this algorithm, ξ determines the number of stages in our procedure. The function *solveModel*($\mathbf{S}$, ModelX) calls ModelX to be solved over the set of schedules $\mathbf{S}$. ModelX can be any of the proposed Models (Model 1, Model 2, Model 3 or Model 4) in the previous chapter. The function *avgCostOf*($\mathbf{S}$) outputs the average of all expected combined costs of the

Algorithm 3.1 Multi-stage method

Input: A set $\mathbf{S}^{\text{init}}$ of initial schedules, the value e that represents the percentage of elite schedules and the value ξ that indicates the maximum number of iterations.

1: $\mathbf{S} = \mathbf{S}^{\text{init}}$
2: $itr = 0$
3: $bestAvgCost = \infty$
4: **while** $itr < \xi$ **do**
5: $\quad solveModel(\mathbf{S}, \text{ModelX})$
6: $\quad$ **if** $avgCostOf(\mathbf{S}) < bestAvgCost$ **then**
7: $\quad\quad \mathbf{S}^{\text{init}} = \mathbf{S}$
8: $\quad\quad bestAvgCost = avgCostOf(\mathbf{S})$
9: $\quad selectElites(\mathbf{S}, e)$
10: $\quad repopulate(\mathbf{S})$
11: $\quad itr = itr + 1$

Output: $\mathbf{S}^{\text{init}}$

baseline schedules in $\mathbf{S}$. The expected combined cost of a baseline schedule $\mathbf{s}$ is the expected combined cost of the best policy that includes $\mathbf{s}$ as its baseline schedule. The function $selectElites(\mathbf{S}, e)$ selects a subset containing e percent of the schedules with the lowest expected combined cost as the subset of elite schedules. Finally, the function $repopulate(\mathbf{S})$ replaces the non-elite schedules with newly generated schedules. Note that the size of the set of schedules ($\mathbf{S}$) always remains the same over all iterations.

We also refer to our multi-stage method as a ξ-stage schedule refinement method. Note that in a 1-stage method no refinement is happening and the input and the output sets of initial schedules are identical.

3.2.2 An alternative initial pool generation scheme

The performance of the initial pool generation scheme (Algorithm 2.2), described in the previous chapter, highly depends on the choice of κ_3. By increasing the value κ_3, the resulting generated schedules are more loose (*i.e.*, include larger buffers). While large buffers may be ideal when reaction costs are high, their presence becomes inefficient when reaction costs are low. Therefore, it might be beneficial to introduce an alternative initial pool generation scheme in which a larger variety of schedules, in terms of looseness, is generated. We provide such an alternative scheme

Algorithm 3.2 An alternative initial pool generation scheme

Input: κ_2 and κ_3

$\mathbf{S}^{\text{init}} = \emptyset$
$n_{\mathbf{S}} = 0$
while $n_{\mathbf{S}} < \kappa_2$ **do**
  if $n_{\mathbf{S}} \bmod 3 = 0$ **then**
    $\hat{\kappa_3} = \kappa_3 - 1$
  else if $n_{\mathbf{S}} \bmod 3 = 1$ **then**
    $\hat{\kappa_3} = \kappa_3$
  else
    $\hat{\kappa_3} = \kappa_3 + 1$
  generate $\hat{\kappa_3}$ random realization $\mathbf{p}_1, ..., \mathbf{p}_{\hat{\kappa_3}}$
  $\mathbf{p}_{\max} = \max\{\mathbf{p}_1, ..., \mathbf{p}_{\hat{\kappa_3}}\}$
  $\mathbf{s} = DH(\mathbf{p}_{\max})$
  $\mathbf{S}^{\text{init}} \leftarrow \mathbf{s}$
  $n_{\mathbf{S}} = n_{\mathbf{S}} + 1$

Output: $\mathbf{S}^{\text{init}}$

in Algorithm 3.2. In the alternative scheme κ_3 is replaced by κ_3'. The idea is to generate a third of the schedules with $\kappa_3' = \kappa_3 - 1$, one third with $\kappa_3' = \kappa_3$ and the last third with $\kappa_3' = \kappa_3 + 1$ ($x \bmod y$ represents the remainder of x divided by y). To indicate which one of the two algorithms (Algorithm 2.2 or Algorithm 3.2) is used to generate the set of initial schedules we introduce a new parameter ϑ which equals 0 if Algorithm 2.2 is used and equals 1 if Algorithm 3.2 is used.

3.2.3 The computational performance

We run Model 3 equipped with our ξ-stage schedule refinement method over the 48 instances introduced in Section 2.3.1. In Table 3.9, an overview of the results is given. In this experiment, we set the problem settings as follows: $w_b = 50, w_r = 100$ and $\lambda = 1$. We also fix some of the algorithm settings: $\kappa_2 = 50, \kappa_3 = 4$ and $\vartheta = 0$. We report the CPU times, the combined cost, the number of states, the number of cuts and the number of continuations for each combination of ξ and κ_1 where $\xi = 1, 2, 5, 10$ or 20 and $\kappa_1 = 500$ or 1000.

Based on the results provided in Table 3.9, we notice that by increasing the number of stages (ξ), the combined cost (CC) is improved consider-

κ_1	ξ	CPU	CC	#states	#cuts	#cons
500	1	94.32	3557.43	2905097	29888	6146
	2	33.29	3536.28	1173225	10303	5737
	5	26.67	3516.02	784955	5698	5568
	10	21.02	3510.65	610524	4608	5556
	20	22.00	3506.13	604575	4591	5519
1000	1	403.90	3547.65	6997712	40172	11780
	2	149.30	3528.60	2790268	13371	11109
	5	111.37	3512.35	1843120	7398	10737
	10	88.27	3507.77	1513340	6328	10725
	20	86.92	3503.70	1435924	6110	10676

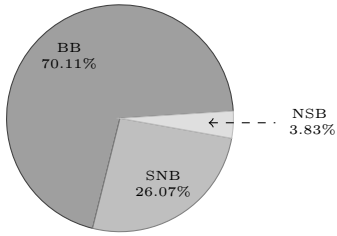
Table 3.9: Summary of the results for Model 3.

ably. Moreover, we notice that having more stages to refine the initial set of schedules, much to our surprise, results in a smaller required total computational time (which also includes the time required to process the schedule refinement stages). To understand this strange behavior, we report detailed CPU times for different sub-procedures in Table 3.10. As expected, the time required to run the initial pool generation procedure (column IPGP) remains the same for all settings, the time required to run ξ-stage procedure (column ξ-Stage) increases by increasing ξ. In contrast to our expectation, the time required to run Model 3 decreases by increasing ξ. This can be justified by the fact that by increasing the number of stages, the schedules in the final set become more similar to one another and thus the numbers of unique cuts and unique continuations are reduced.

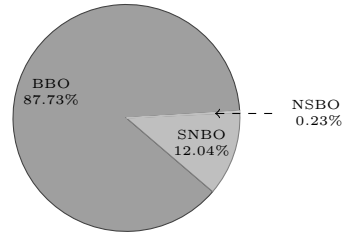
In Figure 3.7, we depict the contributions of three mutually exclusive and collectively exhaustive classes of reactions in the network of Model 3 and in its associated optimal PR-policy when a 10-stage method is used for the problem setting where $w_b = 25$ and $w_r = 0$. Comparing this figure with Figure 3.6, we observe that the percentage of non-selection-based reactions (NSB) in the whole network has been reduced drastically whereas the percentages of buffer-based reactions and selection-based reactions have been increased favorably. We also observe that the contributions of these three classes in their optimal PR-policies have not been changed compared to the contributions of the same classes of reactions in their optimal PR-policies when no schedule refinement technique is implemented. The latest observation supports the hypothesis that the portions of buffer-

κ_1	ξ	IPGP	ξ-Stage	Model	Total
500	1	4.85	0	89.47	94.32
	2	4.18	1.48	27.63	33.29
	5	4.36	1.90	20.40	26.67
	10	4.32	2.48	14.22	21.02
	20	4.30	3.43	14.26	22.00
1000	1	4.16	0	399.72	403.90
	2	4.10	1.44	143.74	149.30
	5	4.13	1.83	105.39	111.37
	10	4.33	2.41	81.51	88.27
	20	4.21	3.49	79.20	86.92

Table 3.10: The detailed CPU times for different sub-procedures.



(a) The average contributions of different classes of reaction in the whole network



(b) The average contributions of different classes of reaction in the optimal PR-policy

Figure 3.7: The average contributions of different classes of reaction when the multi-stage method is used and for the setting where $w_b = 25$ and $w_r = 0$.

based reactions and selection-based reactions in the optimal PR-policy do not significantly change by choosing different sets of schedules.

3.2.4 The choice of parameters

The choices of the parameters may have positive or negative effects on the performance of our methods. In this subsection, we design an experiment to study the effects of seven parameters: w_b, w_r, κ_1, κ_2, κ_3, ϑ and e (note that we deliberately exclude ξ from the analysis in this subsection since its significance is straightforward). The first two parameters are problem

κ_1	κ_2	κ_3	$\vartheta = 0$		$\vartheta = 1$	
			$e = 0.1$	$e = 0.2$	$e = 0.1$	$e = 0.2$
500	50	3	1669.52	1668.71	1671.82	1671.50
		4	1674.58	1673.55	1676.88	1673.88
	100	3	1660.49	1661.30	1661.09	1662.62
		4	1666.42	1666.31	1665.86	1666.20
1000	50	3	1667.93	1666.07	1670.41	1670.12
		4	1673.24	1673.99	1674.50	1674.83
	100	3	1659.84	1661.13	**1659.34**	1660.60
		4	1665.17	1666.10	1664.31	1664.59

Table 3.11: The effect of the different parameters for $w_b = 25$ and $w_r = 0$.

κ_1	κ_2	κ_3	$\vartheta = 0$		$\vartheta = 1$	
			$e = 0.1$	$e = 0.2$	$e = 0.1$	$e = 0.2$
500	50	3	1786.74	1786.39	1785.70	1785.02
		4	1784.12	1783.18	1786.44	1784.78
	100	3	1778.14	1779.01	1779.25	1779.08
		4	1776.42	1777.50	1777.56	1777.13
1000	50	3	1784.45	1785.09	1785.29	1785.04
		4	1783.18	1782.94	1782.74	1782.63
	100	3	1778.02	1776.41	1777.51	1777.16
		4	1776.19	**1775.20**	1776.64	1776.01

Table 3.12: The effect of the different parameters $w_b = 25$ and $w_r = 50$.

setting parameters, whereas the last five parameters are associated with the algorithm setting. These five parameters may influence the performance of our algorithm differently for each combination (w_b, w_r). In this experiment, we allow two different values for each problem setting parameter: $w_b = 25$ or 50 and $w_r = 0$ or 50. Therefore, the total number of combinations of (w_b, w_r) is four: $(25, 0)$, $(25, 50)$, $(50, 0)$ and $(50, 50)$.

In order to study the above-mentioned effects in more detail, for each combination (w_b, w_r) we provide a 2^5 factorial experimental design where $\kappa_1 = 500$ or 1000, $\kappa_2 = 50$ or 100, $\kappa_3 = 3$ or 4, $\vartheta = 0$ or 1 and $e = 0.1$ or 0.2. For each combination of these parameters, we run four replications (by changing the seed of the random generator) of our Model 3 on the benchmark set of 48 instances. The cost associated with each replication is the average of the combined costs of the 48 instances.

κ_1	κ_2	κ_3	$\vartheta = 0$		$\vartheta = 1$	
			$e = 0.1$	$e = 0.2$	$e = 0.1$	$e = 0.2$
500	50	3	3208.95	3211.75	3201.35	3202.97
		4	3241.13	3240.93	3225.94	3228.12
	100	3	3194.38	3193.51	3186.05	3187.41
		4	3220.92	3222.74	3206.17	3205.00
1000	50	3	3207.27	3203.28	3200.15	3199.89
		4	3238.74	3237.65	3223.92	3222.38
	100	3	3190.88	3190.84	**3183.37**	3183.57
		4	3220.65	3218.14	3205.65	3205.64

Table 3.13: The effect of the different parameters $w_b = 50$ and $w_r = 0$.

κ_1	κ_2	κ_3	$\vartheta = 0$		$\vartheta = 1$	
			$e = 0.1$	$e = 0.2$	$e = 0.1$	$e = 0.2$
500	50	3	3390.40	3389.20	3389.05	3392.26
		4	3399.40	3399.06	3397.28	3395.54
	100	3	3376.99	3375.58	3376.55	3377.04
		4	3386.38	3385.22	3382.62	3383.22
1000	50	3	3386.73	3386.36	3385.91	3386.95
		4	3398.50	3397.10	3395.28	3392.92
	100	3	3373.90	**3373.09**	3374.41	3373.78
		4	3384.12	3382.08	3380.00	3381.08

Table 3.14: The effect of the different parameters $w_b = 50$ and $w_r = 50$.

For the problem setting where $w_b = 25$ and $w_r = 0$, Table 3.11 reports the average cost of the four replications for each combination of the five parameters. For this problem setting, the best combination of parameters $(\kappa_1, \kappa_2, \kappa_3, \vartheta, e)$ is $(1000, 100, 3, 1, 0.1)$ (indicated in bold in Table 3.11). For the problem setting $w_b = 25$ and $w_r = 50$ (see Table 3.12), the best combination of parameters is $(1000, 100, 4, 0, 0.2)$. For the problem setting $w_b = 50$ and $w_r = 0$ (see Table 3.13), the best combination of parameters is exactly the same as the best combination for the problem setting where $w_b = 25$ and $w_r = 0$ (*i.e.*, $(1000, 100, 3, 1, 0.1)$). Finally, for the problem setting where $w_b = 50$ and $w_r = 50$ (see Table 3.14), the best combination of parameters is $(1000, 100, 3, 0, 0.2)$ (which is slightly different from the best combination for the problem setting where $w_b = 25$ and $w_r = 50$).

settings	factors				
	κ_1	κ_2	κ_3	ϑ	e
$w_b = 25$ and $w_r = 0$	0.017	0.000	0.000	0.055	0.976
$w_b = 25$ and $w_r = 50$	0.001	0.000	0.000	0.709	0.356
$w_b = 50$ and $w_r = 0$	0.000	0.000	0.000	0.000	0.802
$w_b = 50$ and $w_r = 50$	0.000	0.000	0.000	0.011	0.397

Table 3.15: The associated p-values of the algorithm parameters for different problem settings.

For each problem setting, we perform an ANOVA with main effects and 2-way interactions. Table 3.15 reports the p-values for the main effects for each problem setting. With a significance level of 0.05, we notice that the effects of κ_1, κ_2 and κ_3 are significant for all settings and the effect of e is never significant. The effect of ϑ (the choice of the initial pool generation procedure) is certainly significant when $w_b = 50$, is almost significant when $w_b = 25$ and $w_r = 0$ and is certainly not significant when $w_b = 25$ and $w_r = 50$.

To sum it up, it appears that for all four problem settings, the best combinations include $\kappa_1 = 1000$ and $\kappa_2 = 100$. This is quite in line with our expectation since a larger κ_2 results in a more diverse set of schedules in the initial pool generation scheme and a large κ_1 guarantees more (buffer-based) reaction possibilities. Moreover, when $w_r = 0$ the best value for ϑ is 1 and when $w_r = 50$ the best value for ϑ is 0. This behavior is also inline with our expectation because, as we argued in Section 3.2.2, the presence of large buffers is beneficial when w_r is relatively large and is not beneficial when w_r is relatively small.

3.3 Summary and conclusion

In the first section of this chapter, we introduce two very important classes of reactions, namely the class of selection-based reactions and the class of buffer-based reactions. We show that the class of buffer-based reactions is a sub-class of the class of selection-based reactions and that selection-based reactions, specially those that are also buffer-based, contribute most in the construction of optimal PR-policies. We also show that by relying only on selection-based reactions (or even by relying only on buffer-based reactions), we are still capable of producing high quality PR-policies. In

the second section of this chapter, we introduce a multi-stage refinement technique that helps us build a refined set of initial schedules, thus increasing the probability of having selection-based or buffer-based reactions in the model.

As a future research topic, we strongly advice to study the classes of early start PR-policies and pre-selective PR-policies. Interested researchers might also introduce other classes of PR-policies. Also, we suggest to compare the performance of the best early start (or pre-selective) PR-policy with the best known PR-policy. The analysis of such comparisons and the insights resulting from that will probably open new avenues of research in proactive and reactive project scheduling problems.

Chapter 4

A novel branch and bound algorithm for the chance-constrained resource-constrained project scheduling problem

Everybody [in this country] should learn to program a computer, because it teaches you how to think.

- Steve Jobs

In order to incorporate uncertainty into the *resource-constrained project scheduling problem* (RCPSP) and obtain robust solutions, Lamas and Demeulemeester (2016) introduce the *chance-constrained RCPSP* (CC-RCPSP) in which the makespan is minimized while the actual executed schedule is identical to the planned schedule with a certain minimum probability. In other words, the idea behind CC-RCPSP is to find a proactive solution that is as short as possible and needs no reaction with a certain minimum required probability. This probability, which is usually given

A preliminary version of this chapter appeared as FEB Research Report KBI_1620 at KU Leuven (Davari and Demeulemeester, 2016b). This work has also been submitted for publication.

by the managerial team, will be defined as the *confidence level* in the remainder of this chapter.

In this chapter, we introduce a novel B&B algorithm that can solve instances of the CC-RCPSP until optimality. The novelty of our B&B algorithm is reflected in its branching schemes. In these branching schemes, we use the notion of *casets* that will be introduced and defined in Section 4.3.1. Although the focus of this chapter is to solve the CC-RCPSP, as will be discussed in Section 4.5, the proposed branching schemes can be used to solve any *chance-constrained programming* (CCP) problem with discrete random *right-hand side* (rhs) vector provided that an exact solution oracle exists for the problem's deterministic counterpart.

To the best of our knowledge, the first research on CCP problems with discrete random rhs vector is the one studied by Prékopa (1990). Ruszczyński (2002) proposes a mixed integer programming formulation together with some valid inequalities for a CCP problem with a rhs vector. Recently, stronger facet-defining valid inequalities are proposed by Luedtke et al. (2010) and Küçükyavuz (2012). Among many papers in which a CCP problem with discrete random rhs vector has been introduced to handle uncertainty, we only cite Lamas and Demeulemeester (2016) who propose a B&C algorithm that solves instances of the sample average approximation counterpart of CC-RCPSP to optimality.

The remainder of this chapter is structured as follows. First, in Section 4.1 a compact description of CC-RCPSP and its sample average approximation counterpart (abbreviated by SAA-RCPSP) is given. Then, both a *mixed integer linear programming* (MILP) formulation and a B&B algorithm that optimally solve instances of the SAA-RCPSP are proposed in Section 4.2 and Section 4.3, respectively. Next, computational results are reported in Section 4.4 and the application of the proposed branching schemes for the general integer problem is discussed in Section 4.5. Finally, a summary and conclusion is given in Section 4.6.

4.1 Problem description

Similarly to the previous chapters, we are given a set $N = \{0, 1, ..., n+1\}$ of activities where activities 0 and $n+1$ are the dummy start and dummy end activities. Each activity $i \in N' = N \setminus \{0, n+1\}$ has a stochastic non-negative integer duration $\tilde{p}_i$, with $p_i^{\min} \leq \tilde{p}_i \leq p_i^{\max}$, which follows a discrete distribution $dist(\tilde{p}_i)$. We assume that these stochastic durations are independently distributed. Notice that the durations of the dummy

activities are not stochastic ($\tilde{p}_0 = \tilde{p}_{n+1} = 0$). We are also given a set $\mathcal{R}$ of renewable resource types. Each job i requires r_{ik} units of resource type $k \in \mathcal{R}$ during its processing time and the resource availability of resource type k is denoted by R_k. The set $E \subset \{(i,j)|i,j \in N\}$ defines precedence constraints among the activities where the pair $(i,j) \in E$ indicates that activity j cannot be started before activity i is completed.

Let $\mathbf{s}$, which is a vector of non-negative starting times of the activities, be a solution for the RCPSP and $\hat{\mathbf{p}}$ be its vector of deterministic activity durations. A conceptual formulation for the RCPSP can be formulated as follows:

$$\text{RCPSP: } \min_{\mathbf{s}} s_{n+1}$$

subject to:

$$s_j - s_i \geq \hat{p}_i \qquad \forall (i,j) \in E \quad (4.1)$$

$$\sum_{i \in O_t(\mathbf{s},\hat{\mathbf{p}})} r_{ik} \leq R_k \qquad \forall k \in \mathcal{R}, t = 0, ..., T \quad (4.2)$$

$$\mathbf{s} \in \mathbb{N}^{n+2}. \quad (4.3)$$

where T is an upper bound on the makespan and $O_t(\mathbf{s}, \hat{\mathbf{p}})$ is the corresponding set of ongoing activities at time period $[t, t+1)$ if $\mathbf{s}$ is the vector of starting times and $\hat{\mathbf{p}}$ represents the activity durations. Notice that we assume $\mathbb{N} = \mathbb{Z}^+ \cup \{0\}$. In the above formulation, the objective function is to minimize the completion time of the project (makespan). Constraints (4.1) ensure that all precedence relations among activities are fulfilled, whereas constraints (4.2) represent the resource constraints.

Let $\pi(\cdot)$ be the probability that constraints $\cdot$ are satisfied and $(1-\alpha)$ be the confidence level defined by the decision maker (note that $0 \leq \alpha \leq 1$ and α usually takes a value very close to 0, for example $\alpha = 0.05$, $\alpha = 0.10$ or $\alpha = 0.20$, otherwise the resulting schedule is not feasible for a large number of cases). A conceptual formulation for the CC-RCPSP is given as follows:

$$\text{CC-RCPSP: } \min_{\mathbf{s}} s_{n+1}$$

subject to constraint (4.3) and

$$\pi \left(\begin{array}{ll} s_j - s_i \geq \tilde{p}_i & \forall (i,j) \in E \\ \sum_{i \in O_t(\mathbf{s},\tilde{\mathbf{p}})} r_{ik} \leq R_k & \forall k \in \mathcal{R}, t = 0, ..., T \end{array} \right) \geq 1 - \alpha. \quad (4.4)$$

In the above formulation, constraint (4.4) ensures that the sets of constraints (4.1) and (4.2) combined are not violated with a chance of $(1-\alpha)$.

The CC-RCPSP is proven to be strongly NP-hard following the straightforward reduction from the RCPSP.

4.1.1 A realization-based reformulation

The vector $\tilde{\mathbf{p}} = (\tilde{p}_0, \tilde{p}_1, ..., \tilde{p}_{n+1})$ can be represented by a finite supporting set $\mathfrak{P} = \{\mathbf{p}^1, ..., \mathbf{p}^{|\mathfrak{P}|}\}$ of realizations where each realization $\mathbf{p}^l$ represents a vector of durations $\mathbf{p}^l = (p_0^l, p_1^l, ..., p_{n+1}^l) \in \mathfrak{P}$. Each realization $\mathbf{p}^l$ occurs with a certain probability $\pi(\tilde{\mathbf{p}} = \mathbf{p}^l)$, which is also represented by the much shorter notation π_l. Clearly, the summation of the probabilities of all realizations equals one ($\sum_{\mathbf{p}^l \in \mathfrak{P}} \pi_l = 1$).

Constraint (4.4) is very difficult to tackle. Alternatively, we decide to ensure the feasibility of each solution by finding a *sufficient* subset Y of realizations for which the solution is feasible. A subset Y of realizations is called sufficient if and only if $\sum_{\mathbf{p}^l \in Y} \pi_l \geq (1 - \alpha)$. We introduce $\mathbf{p}^Y$ as the vector of maximum durations of subset Y which is computed as follows:

$$p_i^Y = \max_{\mathbf{p}^l \in Y} \{p_i^l\} \qquad \forall i \in N. \tag{4.5}$$

Let $\mathcal{Y}_{\mathfrak{P}}$ be the set of all sufficient subsets of $\mathfrak{P}$. The CC-RCPSP can be reformulated as follows:

$$\text{CC-RCPSP-R} : \min_{(Y,\mathbf{s})} s_{n+1}$$

subject to constraint (4.3) and

$$s_j - s_i \geq p_i^Y \qquad \forall (i,j) \in E \tag{4.6}$$

$$\sum_{i \in O_t(\mathbf{s},\mathbf{p}^l)} r_{ik} \leq R_k \qquad \forall k \in \mathcal{R}, \mathbf{p}^l \in Y, t = 0, ..., T \tag{4.7}$$

$$\sum_{\mathbf{p}^l \in Y} \pi_l \geq (1 - \alpha) \tag{4.8}$$

$$Y \in \mathcal{Y}_{\mathfrak{P}}. \tag{4.9}$$

In the above formulation, constraints (4.6) make sure that no precedence violation occurs for any realization $\mathbf{p}^l \in Y$ and constraints (4.7) ensure that no resource violation occurs for any realization $\mathbf{p}^l \in Y$. Constraint (4.8) dictates a confidence level of at least $(1 - \alpha)$.

Notice that constraints (4.7) are not linear and therefore we propose the following overprotecting but linear constraints to replace them:

$$\sum_{i \in O_t(\mathbf{s}, \mathbf{p}^Y)} r_{ik} \leq R_k \qquad \forall k \in \mathcal{R}, t = 0, ..., T. \quad (4.10)$$

The following formulation results:

$$\text{CC-RCPSP-R-OP} : \min_{(Y, \mathbf{s})} s_{n+1}$$

subject to constraints (4.3), (4.6), (4.8)-(4.10).

Although CC-RCPSP-R-OP is an overprotected formulation, it can be used as an intermediate step to obtain a sample average approximation formulation.

4.1.2 A sample average approximation

The size of the associated finite supporting set of realizations is often too large, and thus we use a *sample average approximation* (SAA) technique, which is based on Monte Carlo sampling, to generate a much smaller set $\hat{\mathfrak{P}}$ of m realizations which approximates the original set $\mathfrak{P}$ (note that the size m influences the quality of the approximation).

We introduce the associated SAA counterpart of the CC-RCPSP (in short SAA-RCPSP) as follows:

$$\text{SAA-RCPSP} : \min_{(Y, \mathbf{s})} s_{n+1}$$

subject to constraint (4.3), (4.6), (4.10) and

$$\sum_{\mathbf{p}^l \in Y} \pi_l \geq (1 - \hat{\alpha}) \quad (4.11)$$

$$Y \in \mathcal{Y}_{\hat{\mathfrak{P}}}. \quad (4.12)$$

where $1 - \hat{\alpha}$ is the required confidence level for this counterpart problem (note that $1 - \hat{\alpha} > 1 - \alpha$). In the special case where $\pi_l = 1/m$ for all realizations $\mathbf{p}^l$ in $\hat{\mathfrak{P}}$, constraint (4.11) can be replaced by constraint (4.13).

$$|Y| \geq \lceil (1 - \hat{\alpha}) \times m \rceil \quad (4.13)$$

Let $\epsilon = \alpha - \hat{\alpha}$ and let $p^{\max} = \max_{i \in N} \{p_i^{\max}\}$. Following the theorems in Luedtke and Ahmed (2008), one can show that any feasible solution to an instance of SAA-RCPSP is also feasible to the associated instance of CC-RCPSP with a probability of $(1 - \theta)$ if

$$m \geq \frac{1}{2\epsilon^2} \log\left(\frac{1}{\theta}\right) + \frac{n}{2\epsilon^2} \log(p^{\max}). \quad (4.14)$$

For instance if $p^{\max} = 30$, then m must be at least 9123 to ensure that any feasible solution to an instance of SAA-RCPSP is also feasible to the associated instance of CC-RCPSP with a probability of 0.95. However, Luedtke and Ahmed (2008) also argue that this lower bound for m is very conservative and one often achieves similar confidence with much smaller m. In this chapter, we choose m between 100 and 1600.

As the final part of this section, we pinpoint two remarks. First, because solving CC-RCPSP becomes computationally intractable, in Sections 4.2 and 4.3, we opt to solve the SAA-RCPSP, which is proven to be a good approximation for CC-RCPSP (Lamas and Demeulemeester, 2016). However, given unlimited computational resources, the methods presented in Sections 4.2 and 4.3 can solve instances of CC-RCPSP. Second, while using the SAA technique to generate a set of realizations, generally the probability of occurrence of every single generated realization is the same as that of any other realization in that set and equals $1/m$. However, our methods are designed for a more general case where the probabilities of occurrences of realizations need not be the same.

4.2 A mathematical formulation

In this section, we introduce a MILP formulation for the SAA-RCPSP which is the chance-constrained version of the resource-flow formulation proposed by Artigues et al. (2003). Let us define variables x_{ij} that equal one if activity i is completed before the start of activity j and zero otherwise. We also introduce the variables f_{ijk} which represent the amount of resource type k that is passed from activity i to activity j. Finally, we define variables y_l such that if $y_l = 1$, then $\mathbf{s}$ must be feasible for realization $\mathbf{p}^l$ ($\mathbf{p}^l \in Y$) and if $y_l = 0$, then the feasibility of $\mathbf{s}$ for realization $\mathbf{p}^l$ is not necessary ($\mathbf{p}^l \notin Y$). We propose the following MILP formulation for the SAA-RCPSP:

$$\text{SAA-RCPSP-MILP} : \min_{\mathbf{s}} s_{n+1}$$

subject to:

$$x_{ij} = 1 \qquad \forall (i,j) \in E \tag{4.15}$$

$$x_{ij} + x_{ji} \leq 1 \qquad \forall (i,j) \in N^2, i \neq j \tag{4.16}$$

$$f_{ijk} \leq M^r x_{ij} \qquad \forall (i,j) \in N'^2, i \neq j, \forall k \in \mathcal{R} \tag{4.17}$$

$$f_{0jk} \leq r_{jk} x_{0j} \qquad \forall (0,j) \in N'^2, j \neq 0, \forall k \in \mathcal{R} \tag{4.18}$$

$$f_{i(n+1)k} \leq r_{ik} x_{i(n+1)} \qquad \forall (i, n+1) \in N'^2, i \neq n+1, \forall k \in \mathcal{R} \tag{4.19}$$

$$\sum_{j \in N} f_{0jk} = R_k \qquad \forall k \in \mathcal{R} \tag{4.20}$$

$$\sum_{j \in N} f_{ijk} = r_{ik} \qquad \forall i \in N, \forall k \in \mathcal{R} \tag{4.21}$$

$$\sum_{j \in N} f_{jik} = r_{ik} \qquad \forall i \in N, \forall k \in \mathcal{R} \tag{4.22}$$

$$s_j - s_i - M^p(x_{ij} - 1) \geq y_l p_i^l \qquad \forall (i, j) \in N^2, i \neq j, \forall l = 1, ..., m \tag{4.23}$$

$$\sum_{l=1}^{m} (1 - y_l)\pi_l \leq \hat{\alpha} \tag{4.24}$$

$$\mathbf{s} \in \mathbb{N}^{n+2}, \mathbf{x} \in \{0,1\}^{(n+2)^2} \tag{4.25}$$

$$\mathbf{f} \in \mathbb{R}^{|\mathcal{R}|(n+2)^2}, \mathbf{y} \in \{0,1\}^m \tag{4.26}$$

where $M^p = \sum_{i \in N} p_i^{\max}$ and $M^r = \min\{r_{ik}, r_{jk}\}$. In the above formulation, the set of constraints (4.15) enforces the precedence relations among activities. The set of constraints (4.16) ensures that the two variables x_{ij} or x_{ji} are not both one. The sets of constraints (4.17)-(4.22) represent the resource flow among activities. Constraints (4.23) guarantee that $\mathbf{s}$ is feasible for realization $\mathbf{p}^l$ if $y_l = 1$. Finally, constraint (4.24) ensures that the cumulative probability of the occurrences of the realizations in the set Y (realizations $\mathbf{p}^l$ for which $y_l = 1$) is at least $(1 - \hat{\alpha})$. Based on the definitions, for each feasible vector $\mathbf{y}$, we can derive an associated sufficient set of realizations Y. Let us introduce $\mathbf{p}^{\mathbf{y}}$ which equals its associated vector $\mathbf{p}^Y$. We have

$$p_i^Y = p_i^{\mathbf{y}} = \max_{\mathbf{p}^l \in \hat{\mathfrak{P}}} \{y_l p_i^l\} = \max_{\mathbf{p}^l \in Y} \{p_i^l\}.$$

The set of constraints (4.23) can be replaced by the following set of conceptual constraints:

$$s_j - s_i - M^p(x_{ij} - 1) \geq p_i^{\mathbf{y}} \qquad \forall (i, j) \in N^2, i \neq j. \tag{4.27}$$

Although constraints (4.27) form a nonlinear term, they can be used to understand which realizations should be included in or excluded from Y. An obvious finding is that in order to reduce the makespan (s_{n+1}) and possibly find an optimal solution, we must choose $\mathbf{y}$ in such a way that

for some activities i, the value $p_i^{\mathbf{y}}$ takes a smaller value than $p_i^{\max}$. This finding is the main motivation for the reformulation introduced in the following subsection.

4.2.1 A stronger formulation

In Luedtke et al. (2010), a stronger reformulation has been proposed for the general chance-constrained optimization problem. We follow the steps to obtain a stronger reformulation for the SAA-RCPSP. Let $\boldsymbol{\delta}_i$ be the vector of activity durations for each realization in $\hat{\mathfrak{P}}$, sorted in non-increasing order. In other words, δ_i^k is the kth largest duration for activity i. In order to keep track of the realizations in each sorted vector $\boldsymbol{\delta}_i$, we introduce σ_{ik} which represents the associated realization for each pair (i,k) (in other words, we have $\delta_i^k = p_i^{\sigma_{ik}}$). For example, if p_i^5 is the second largest duration for activity i, then $\delta_i^2 = p_i^5$ and $\sigma_{i2} = 5$. We provide a more detailed example in Section 4.2.2. For each activity i, we define $\eta_i \leq m$ as the largest $k \in \{1, ..., m\}$ that satisfies $\sum_{s=1}^{k} \pi_{\sigma_{is}} \leq \hat{\alpha}$. Parameter η_i can be described as an upper bound on the number of realizations $\mathbf{p}^l$ that can be excluded from Y ($y_l = 0$) such that the resulting Y is still sufficient and $p_i^{\mathbf{y}}$ is minimized. Notice that $\eta_0 = \eta_{n+1} = 0$. Thus, a stronger formulation can be obtained for the SAA-RCPSP:

SAA-RCPSP-MILP2: $\min_{\mathbf{s}} s_{n+1}$

subject to: constraints (4.15)-(4.22), (4.24)-(4.26) and

$$s_j - s_i - M^p(x_{ij} - 1) \geq \delta_i^1 - \sum_{k=1}^{\eta_i} (\delta_i^k - \delta_i^{k+1})(1 - z_{ik}) \quad \forall (i,j) \in N^2, i \neq j \tag{4.28}$$

$$z_{ik} - y_{\sigma_{ik}} \geq 0 \qquad \forall i \in N, k = 1, ..., \eta_i \tag{4.29}$$

$$z_{i,k+1} - z_{ik} \geq 0 \qquad \forall i \in N, k = 1, ..., \eta_i \tag{4.30}$$

$$z_{i,\eta_i+1} = 1 \qquad \forall i \in N \tag{4.31}$$

$$z_{ik} \in \{0,1\} \qquad \forall i \in N, k = 1, ..., \eta_i. \tag{4.32}$$

In the above formulation, z_{ik} is zero if we decide not to consider the kth largest duration for activity i and one otherwise. Also if $z_{ik} = 0$, then the feasibility of the resulting schedule is not guaranteed for its associated realization (σ_{ik}) and thus we enforce $y_{\sigma_{ik}} = 0$ (constraints (4.29)). Obviously, it would be inefficient if $z_{ik} = 1$ and $z_{i,k+1} = 0$. Therefore, such

i	p_i^1	p_i^2	p_i^3	p_i^4	p_i^5	p_i^6	p_i^7	p_i^8	p_i^9	p_i^{10}
1	3	3	3	2	2	**3**	3	1	1	3
2	10	5	9	6	8	**6**	7	11	11	6
3	2	3	4	3	3	**2**	5	4	5	1
4	4	3	6	6	6	**5**	4	2	4	4
5	7	7	5	9	12	**6**	6	5	9	9
6	7	9	4	6	5	**4**	9	7	9	8
7	3	4	4	2	5	**4**	6	4	6	2
8	1	1	3	3	2	**3**	3	3	2	2

Table 4.1: The set $\hat{\mathfrak{P}}$ of realizations for the example.

i	δ_i^1	δ_i^2	δ_i^3	δ_i^4	δ_i^5	δ_i^6	δ_i^7	δ_i^8	δ_i^9	δ_i^{10}
1	3	3	3	**3**	3	3	2	2	1	1
2	11	11	10	9	8	7	6	**6**	6	5
3	5	5	4	4	3	3	3	2	**2**	1
4	6	6	6	**5**	4	4	4	4	3	2
5	12	9	9	9	7	7	**6**	6	5	5
6	9	9	9	8	7	7	6	5	4	**4**
7	6	6	5	4	4	**4**	4	3	2	2
8	3	3	**3**	3	3	2	2	2	1	1

Table 4.2: The matrix $\boldsymbol{\delta}$ for the example.

cases are eliminated by constraints (4.30). It is not very difficult to see that in no feasible schedule z_{i,η_i+1} could be equal to zero and therefore the strength of the linear relaxation bound can be improved by adding constraints (4.31). Notice that the set of constraints (4.28)-(4.32) is a much stronger alternative for the set of constraints (4.23).

4.2.2 An example

We consider an instance of the problem with $n = 8$ and $m = 10$. The precedence relations among activities as well as the resource consumptions are given in Figure 3.1. An example set of realizations $\hat{\mathfrak{P}}$ and the associated matrices $\boldsymbol{\delta}$ and $\boldsymbol{\sigma}$ are given in Tables 4.1 to 4.3.

In these three tables, the numbers associated with realization $\mathbf{p}^6$ are shown in bold. Note that these tables will be used in all examples given in the remainder of this chapter.

i	$\sigma_{i,1}$	$\sigma_{i,2}$	$\sigma_{i,3}$	$\sigma_{i,4}$	$\sigma_{i,5}$	$\sigma_{i,6}$	$\sigma_{i,7}$	$\sigma_{i,8}$	$\sigma_{i,9}$	$\sigma_{i,10}$
1	1	2	3	**6**	7	10	4	5	8	9
2	8	9	1	3	5	7	4	**6**	10	2
3	7	9	3	8	2	4	5	1	**6**	10
4	3	4	5	**6**	1	7	9	10	2	8
5	5	4	9	10	1	2	**6**	7	3	8
6	2	7	9	10	1	8	4	5	3	**6**
7	7	9	5	2	3	**6**	8	1	4	10
8	3	4	**6**	7	8	5	9	10	1	2

Table 4.3: The matrix $\boldsymbol{\sigma}$ for the example.

4.3 Branch-and-bound

In this section, we propose a B&B algorithm that solves the SAA-RCPSP. The idea is to find a schedule with minimum makespan that is feasible for at least one sufficient set of realizations. Let Y denote a set of realizations and let $\mathcal{O}(\mathbf{p}^Y)$ be an optimization oracle that solves the RCPSP while the vector of activity durations is $\mathbf{p}^Y$. We define $\mathbf{s}^Y$ as the schedule obtained by running $\mathcal{O}(\mathbf{p}^Y)$. Because $\mathbf{s}^Y$ is feasible for the problem with $\mathbf{p}^Y$, it is also feasible for all realizations in Y. Thus, if Y is a sufficient subset, then it provides a confidence level of $(1 - \hat{\alpha})$.

For each Y, there exists a complement set $\bar{Y}$ of realizations such that $Y \cup \bar{Y} = \hat{\mathfrak{P}}$. For each pair $(Y, \bar{Y})$, Y is referred to as the *included set* and $\bar{Y}$ is referred to as the *excluded set*. Obviously, if Y is a sufficient subset, then the cumulative probability of occurrence of the realizations in $\bar{Y}$ is smaller than or equal to $\hat{\alpha}$. Let Ξ be the set of all pairs $(Y, \bar{Y})$ for which $\sum_{\mathbf{p}^l \in \bar{Y}} \pi_l \leq \hat{\alpha}$. The SAA-RCPSP can be reformulated as follows:

$$\min_{(Y,\bar{Y}) \in \Xi} s_{n+1}^Y.$$

Inspired by this conceptual formulation, we aim to use a B&B algorithm to enumerate all pairs $(Y, \bar{Y}) \in \Xi$ and find a pair $(Y^*, \bar{Y}^*)$ with the minimum corresponding makespan $(s_{n+1}^{Y^*})$.

4.3.1 Constructing the tree

As we already mentioned, we aim to find the optimal pair $(Y^*, \bar{Y}^*)$ using a B&B algorithm. Since the included set (Y) and the excluded set $(\bar{Y})$

are the complement of each other, it is sufficient to only enumerate all valid excluded sets (or all valid included sets). A conventional branching scheme can be perfectly used to enumerate all excluded sets by starting from the set of all realizations and in each node/level excluding a single realization. However, we opt not to directly exclude single realizations, but instead we exclude *chained sets* (abbreviated to casets) of realizations in our novel branching schemes.

Let us introduce C_i^k as the *caset* of realizations with the k^{th} highest duration for activity i. The exclusion of caset C_i^k from pair $(Y, \bar{Y}) \in \Xi$ results in the pair $(Y \setminus C_i^k, \bar{Y} \cup C_i^k)$. This exclusion is *possible* only if the resulting pair $(Y \setminus C_i^k, \bar{Y} \cup C_i^k)$ is also a member of Ξ.

We immediately notice that not all possible exclusions are necessarily evaluated. Only those exclusions that have a positive impact must be considered. Therefore, we introduce the notion of *beneficial* exclusions. The exclusion of caset C_i^k from pair $(Y, \bar{Y}) \in \Xi$ is labeled beneficial if and only if it is possible and $\mathbf{p}^{Y \setminus C_i^k} < \mathbf{p}^Y$ (obviously if $\mathbf{p}^{Y \setminus C_i^k} = \mathbf{p}^Y$, then $\mathbf{s}^{Y \setminus C_i^k} = \mathbf{s}^Y$ and the associated exclusion is not beneficial). We assume that $\mathbf{p} < \mathbf{p}'$ if $\exists i \in N, p_i < p'_i$ and $\forall i' \in N \setminus \{i\}, p_i \leq p'_i$.

Corollary 4.1. *The exclusion of caset C_i^k from pair $(Y, \bar{Y}) \in \Xi$ is beneficial if it is possible, $\bar{Y} \cup C_i^k \neq \bar{Y}$ and $(C_i^1 \cup ... \cup C_i^{k-1}) \subseteq \bar{Y}$. The reverse relation does not necessarily hold.*

Some casets can never be beneficially excluded. Since considering such casets is not efficient, we limit our search to only *eligible* casets.

Definition 4.1 (Eligible caset)**.** *An eligible caset is a caset that can be beneficially excluded.*

We introduce set $\mathbf{C}^E$ as the set of all eligible casets. Let us compute $\pi(C_i^k) = \sum_{\mathbf{p}^l \in C_i^k} \pi_l$. The following theorem is derived.

Theorem 4.1. *A caset C_i^k is eligible if and only if $\sum_{s=1}^{k} \pi(C_i^s) \leq \hat{\alpha}$.*

Proof. Consider the following two sets:

$$Y_1 = \hat{\mathfrak{P}} \setminus (C_i^1 \cup ... \cup C_i^{k-1}) \text{ and } Y_2 = Y_1 \setminus C_i^k.$$

If the inequality $\sum_{s=1}^{k} \pi(C_i^s) \leq \hat{\alpha}$ holds, both sets must be sufficient sets. We also know that $p_i^{Y_2} < p_i^{Y_1}$. Therefore, we conclude that the exclusion of C_i^k from Y_1 is beneficial and C_i^k is eligible.

These sets are named chained sets because they often relate to each other like a group of chains

On the other hand, if C_i^k is eligible, it can be beneficially excluded and therefore $(C_i^1 \cup ... \cup C_i^{k-1}) \subseteq \bar{Y}$. Since any beneficial exclusion is possible, we have $\sum_{s=1}^{k} \pi(C_i^s) \leq \hat{\alpha}$. □

We define ζ_i as the number of eligible casets associated with activity i. For each activity i, the caset corresponding to the highest duration (C_i^1) is referred to as the *lead caset*. Other casets that associate with the second duration, third duration, etc are referred to as the *second caset, third caset, etc*, respectively.

Example 4.1. *Let* $\hat{\alpha} = 0.4$ *and*

$$\boldsymbol{\pi} = (\pi_1, ..., \pi_{10}) = (0.2, 0.15, 0.15, 0.1, 0.1, 0.1, 0.05, 0.05, 0.05, 0.05).$$

The set of eligible casets for the example is

$$\begin{aligned}\mathbf{C}^E = \{&C_2^1 = \{8,9\}, C_2^2 = \{1\},\\ &C_3^1 = \{7,9\}, C_3^2 = \{3,8\},\\ &C_4^1 = \{3,4,5\},\\ &C_5^1 = \{5\}, C_5^2 = \{4,9,10\},\\ &C_6^1 = \{2,7,9\}, C_6^2 = \{10\},\\ &C_7^1 = \{7,9\}, C_7^2 = \{5\}\}.\end{aligned}$$

In this case, the lead casets are $C_2^1, C_3^1, C_4^1, C_5^1, C_6^1$ *and* C_7^1*. We also compute:* $\zeta_0 = \zeta_1 = \zeta_8 = \zeta_9 = 0$*,* $\zeta_4 = 1$ *and* $\zeta_2 = \zeta_3 = \zeta_5 = \zeta_6 = \zeta_7 = 2$*.*

4.3.1.1 Constructing the activity list

In this part, we construct an *activity list* (AL) (note that all activities i for which $\zeta_i = 0$ are not considered in this list) that defines the order based on which activities are considered in our proposed branching schemes (that are introduced in Sections 4.3.1.2 and 4.3.1.3). We introduce different *priority rules* that can be used to construct such an AL. Each priority rule consists of a *sorting criterion* based on which the activities are sorted and (possibly) a number of tie breakers. Two example priority rules are given below.

(Rule 1) We sort activities based on the lexicographical order of the realizations in their σ_i vector.

(Rule 2) We sort activities based on the decreasing order of their lead caset sizes. As the first/second/etc tie breaking rule, we consider the sizes of their second/third/etc casets.

Although the performance of these two priority rules are acceptable (see Section 4.4.2), better rules can be achieved by incorporating more important criteria. We propose to take three criteria into consideration while sorting the activities:

- The first criterion is the *total slack* (TS) of the activity. Given realization $\mathbf{p}$ and a feasible schedule $\mathbf{s}$, let $es_i(\mathbf{s},\mathbf{p})$ and $ls_i(\mathbf{s},\mathbf{p})$ be the earliest and the latest starting times of activity i, respectively. We denote by $\varepsilon_i(\mathbf{s},\mathbf{p})$ the total slack of activity i for the given pair $(\mathbf{s},\mathbf{p})$. This total slack is computed as follows: $\varepsilon_i(\mathbf{s},\mathbf{p}) = ls_i(\mathbf{s},\mathbf{p}) - es_i(\mathbf{s},\mathbf{p})$. Activities with smaller total slack values are favored to be positioned first in the list. In our experiments, we use $\mathbf{s}^{\hat{\mathfrak{P}}}$ and $\mathbf{p}^{\hat{\mathfrak{P}}}$ to compute total slacks.

- The second criterion is the number of eligible casets (NEC) associated with the activity. Those activities with smaller numbers of casets (smaller ζ_i) are favored to be positioned first in the list.

- The last criterion is the *influence factor* (IF) of the activity. The influence factor of activity i, which is denoted by ψ_i, is computed as follows:

$$\psi_i = \sum_{k=1}^{\zeta_i} \frac{p_i^{\max} - v(C_i^k)}{\sum_{s=1}^{k} |C_i^s|}$$

  where $v(C_i^k)$ represents the resulting duration of activity i if C_i^k and its corresponding earlier casets (C_i^{k-1},C_i^{k-2},etc) are eliminated. Activities with larger influence factor values are favored to be positioned first in the list.

Example 4.2. *We compute the IF for activity* 5 *as follows:*

$$\psi_5 = \frac{12-9}{1} + \frac{12-7}{4} = 4.25.$$

One of these three criteria is selected as the main sorting criterion. The other two criteria are exploited as the first and the second tie breakers. The question is in which order we consider these three criteria such

Priority rule	Sorting criterion	First tie breaker	Second tie breaker
Rule 3	TS (↑)	NEC (↑)	IF (↓)
Rule 4	TS (↑)	IF (↓)	NEC (↑)
Rule 5	NEC (↑)	TS (↑)	IF (↓)
Rule 6	NEC (↑)	IF (↓)	TS (↑)
Rule 7	IF (↓)	TS (↑)	NEC (↑)
Rule 8	IF (↓)	NEC (↑)	TS (↑)

Table 4.4: Different priority rules obtained by different combinations of the following three criteria: total slack (TS), number of casets (NEC) and influence factor (IF). Symbol (↑) represents an ascending order and (↓) denotes a descending order.

that the performance of the branch-and-bound algorithm is maximized. These criteria can be ordered in six different ways. Table 4.4 depicts these six different ways, each associated with a different priority rule (Rule 3, ..., Rule 8). Notice that all priority rules (Rule 1 − Rule 8) share a final tie breaking rule which dictates the activity with the smaller index to be positioned first.

4.3.1.2 Branching scheme 1

The nodes in our B&B are denoted by $\mathcal{N}_u$ where u is the index of the node, indicating the sequence in which the nodes are visited. In each node $\mathcal{N}_u$ (except in the root node), we decide to exclude a caset $C_i^k \in \mathbf{C}^E$ that is referred to as the *target caset*. The target caset of node $\mathcal{N}_u$ is denoted by $\Theta(\mathcal{N}_u)$. The *direct father* of a node $\mathcal{N}_u$ is the node from which it branched whereas a node's *transitive father* is an ancestor (*i.e.*, father of the father (grandfather), father of the grandfather, etc) of the node. A node's set of excluded casets is the set of all target casets of the node, its direct father and all of its transitive fathers. Since each node has a one-to-one correspondence with its set of excluded casets, without loss of generality, we let both the node and its associated set of excluded casets be represented by the same notation $\mathcal{N}_u$. Excluding a caset is equivalent to the exclusion of all its realizations. For each node $\mathcal{N}_u$, the pair $(Y^{\mathcal{N}_u}, \bar{Y}^{\mathcal{N}_u})$ is the node's associated pair in Ξ. Notice that a caset may be considered excluded, before being excluded itself, with the exclusion of a combination of some other casets.

Example 4.3. *For this example,* $\mathcal{N}_0 = \emptyset$ *represents the root node,* $\mathcal{N}_1 = \{C_6^1\}$*, which is branched from* $\mathcal{N}_0$*, is the node where only* C_6^1 *is excluded and* $\mathcal{N}_2 = \{C_6^1, C_6^2\}$*, which is branched from* $\mathcal{N}_1$*, represents the node where both* C_6^1 *and* C_6^2 *are excluded. The root node is the father of* $\mathcal{N}_1$ *and the only transitive father of* $\mathcal{N}_2$*. The target casets are* $\Theta(\mathcal{N}_1) = C_6^1$ *and* $\Theta(\mathcal{N}_2) = C_6^2$ *for* $\mathcal{N}_1$ *and* $\mathcal{N}_2$*, respectively. Also,*

$$Y^{\mathcal{N}_2} = \{\mathbf{p}^1, \mathbf{p}^3, \mathbf{p}^4, \mathbf{p}^5, \mathbf{p}^6, \mathbf{p}^8\} \text{ and}$$
$$\bar{Y}^{\mathcal{N}_2} = \{\mathbf{p}^2, \mathbf{p}^7, \mathbf{p}^9, \mathbf{p}^{10}\}.$$

Each node $\mathcal{N}_u$ corresponds with a set $D(\mathcal{N}_u)$ of *effective casets*. For $\mathcal{N}_u$, a caset C is an effective caset if its exclusion from $\mathcal{N}_u$ is beneficial and its associated activity is positioned after the associated activity of $\Theta(\mathcal{N}_u)$. The former condition guarantees an improvement in the child's vector of durations whereas the latter condition prevents duplicate exclusions of casets. Since it is not efficient to exclude any ineffective caset, the target casets of the children of a node must be members of its set of effective casets.

Example 4.4. *The set of effective casets for the root node and for priority rule Rule 1 consists of all lead casets (*$D(\mathcal{N}_0) = \{C_6^1, C_4^1, C_5^1, C_3^1, C_7^1, C_2^1\}$*) and therefore no child of the root node has a non-lead target caset. For the node* $\mathcal{N}_2 = \{C_6^1, C_6^2\}$ *we have:*

$$D(\mathcal{N}_2) = \{C_5^1, C_7^2, C_2^1\}.$$

Note that C_3^1 *and* C_7^1 *are both subsets of* $\bar{Y}^{\mathcal{N}_2}$ *and thus the exclusions of* C_3^1 *and* C_7^1 *from* $(Y^{\mathcal{N}_2}, \bar{Y}^{\mathcal{N}_2})$ *are not beneficial. Additionally, the exclusions of* C_3^2 *and* C_4^1 *from* $(Y^{\mathcal{N}_2}, \bar{Y}^{\mathcal{N}_2})$ *are not possible. Since all effective casets must be both beneficial and possible,* $D(\mathcal{N}_2)$ *only consists of* C_5^1, C_7^2 *and* C_2^1*. With similar considerations, we have:* $D(\mathcal{N}_4) = D(\mathcal{N}_5) = D(\mathcal{N}_6) = \emptyset$*.*

The branching starts with the *root* node ($\mathcal{N}_0$). The root node, which corresponds with the situation where no caset has been excluded ($\mathcal{N}_0 = \emptyset$), is branched into a number of child nodes, each corresponding with the exclusion of a certain caset (remember that this caset must be a member of $D(\mathcal{N}_0)$ and therefore should be both possible and beneficial). Each of these child nodes is then branched into its own children and so on. Backtracking happens in a node if all its children have already been visited or if its set of eligible casets is an empty set. Each node in this B&B tree is

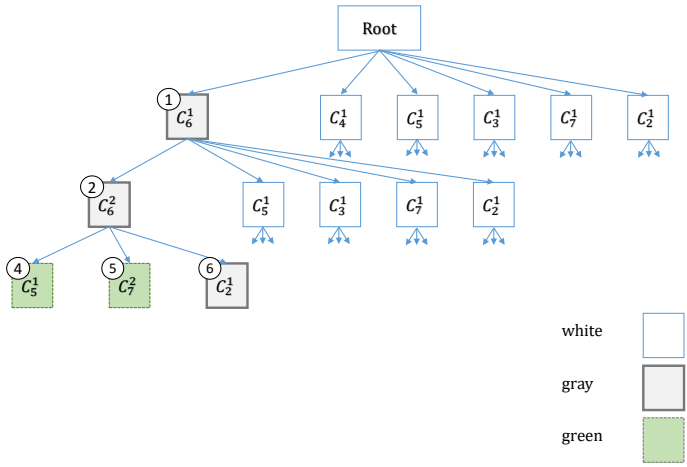


Figure 4.1: Branching scheme 1.

associated with a feasible solution for SAA-RCPSP. Thus, $\mathrm{UB}^{\mathcal{N}_u} = s_{n+1}^{Y^{\mathcal{N}_u}}$ is an upper bound for the SAA-RCPSP. We denote the best upper bound found so far by UB^*.

Although we avoid duplicated combinations of excluded casets by introducing the set of effective casets, it is very difficult to modify the branching scheme such that no duplicated combination of excluded realizations occurs. This difficulty stems from the fact that many casets may contain one or more common realizations. In order to clarify this branching scheme, we provide an example in which the B&B tree is searched in depth-first mode.

Example 4.5. *Figure 4.1 depicts a part of the B&B tree where branching scheme 1 is used in a depth-first mode. Each node is represented by a square. Since all information of a caset cannot be printed for each node (because of limited space), only its target caset is printed. Also, due to lack of space, the tree is not complete (the nodes with white background have not been continued and the nodes with colored background have been continued).*

The root node is branched into nodes with effective target casets $C_6^1, C_4^1, C_5^1, C_3^1, C_7^1$ *and* C_2^1. *Among the children of the root node, node* $\mathcal{N}_1 = \{C_6^1\}$ *is branched first since its target caset's associated activity is positioned earlier in the AL (which is constructed according priority rule Rule 1 for this example). Then among the children of* $\mathcal{N}_1 = \{C_6^1\}$, *node* $\mathcal{N}_2 = \{C_6^1, C_6^2\}$ *is branched first and so on.*

All gray nodes are those for which $\sum_{\mathbf{p}_l \in \bar{Y}^{\mathcal{N}}} \pi_l < \hat{\alpha}$ *and all green nodes are those for which* $\sum_{\mathbf{p}_l \in \bar{Y}^{\mathcal{N}}} \pi_l = \hat{\alpha}$. *For example, consider node* $\mathcal{N}_2 = \{C_6^1, C_6^2\}$. *For this node* $\bar{Y}^{\mathcal{N}_2} = \{\mathbf{p}^2, \mathbf{p}^7, \mathbf{p}^9, \mathbf{p}^{10}\}$ *and* $\sum_{\mathbf{p}^l \in \bar{Y}^{\mathcal{N}_2}} \pi_l = 0.3 < \hat{\alpha}$ *(where* $\hat{\alpha} = 0.4$*), therefore its background color is gray. For node* $\mathcal{N}_5 = \{C_6^1, C_6^2, C_7^2\}$, $\bar{Y}^{\mathcal{N}_5} = \{\mathbf{p}^2, \mathbf{p}^5, \mathbf{p}^7, \mathbf{p}^9, \mathbf{p}^{10}\}$ *and* $\sum_{\mathbf{p}^l \in \bar{Y}^{\mathcal{N}_5}} \pi_l = 0.4 = \hat{\alpha}$, *therefore its background color is green. We also compute*

$$\mathbf{s}^{Y^{\mathcal{N}_2}} = (0, 0, 3, 5, 0, 9, 9, 14, 19, 22) \rightarrow \mathrm{UB}^{\mathcal{N}_5} = 22 \text{ and}$$
$$\mathbf{s}^{Y^{\mathcal{N}_5}} = (0, 0, 0, 8, 3, 12, 11, 12, 18, 21) \rightarrow \mathrm{UB}^{\mathcal{N}_6} = 21.$$

Some of the nodes can be dominated by computing a lower bound. In each node $\mathcal{N}_u$, we compute a lower bound, that is denoted by $\mathrm{LB}^{\mathcal{N}_u}$. This lower bound is computed as explained in the following steps:

1. We construct $\hat{\mathcal{N}}_u$ which represents an extremely pessimistic case. $\hat{\mathcal{N}}_u$ initially contains all casets in $\mathcal{N}_u$.

2. Let $C_i^k = \Theta(\mathcal{N}_u)$. Add all casets $C_i^{k_1}$ with $k_1 = k+1, ..., \kappa_i$ to $\hat{\mathcal{N}}_u$ where κ_i is the largest integer that satisfies the following condition:

$$\sum_{\mathbf{p}_l \in \bar{Y}^{\mathcal{N}_u} \cup \{C_i^{k+1}, ..., C_i^{\kappa_i}\}} \pi_l \leq \hat{\alpha}.$$

3. Let activity i be the activity associated with the current node. For each activity $i' \in N$ that is positioned after activity i in the AL, add all casets $C_{i'}^{k_1}$ with $k_1 = 1, ..., \kappa_{i'}$ to $\hat{\mathcal{N}}_u$ where $\kappa_{i'}$ is the largest integer that satisfies the following condition:

$$\sum_{\mathbf{p}_l \in \bar{Y}^{\mathcal{N}_u} \cup \{C_{i'}^1, ..., C_{i'}^{\kappa_{i'}}\}} \pi_l \leq \hat{\alpha}.$$

4. $\mathrm{LB}^{\mathcal{N}_u} = s_{n+1}^{Y^{\hat{\mathcal{N}}_u}}$.

Every node $\mathcal{N}_u$ in our B&B tree is dominated if $\text{LB}^{\mathcal{N}_u} \geq \text{UB}^*$. Notice that this dominance rule is not considered in the example tree presented in Figure 4.1.

Example 4.6. *Consider node* $\mathcal{N}_2 = \{C_6^1, C_6^2\}$ *in Figure 4.1. We have:* $\bar{Y}^{\mathcal{N}_2} = \{\mathbf{p}^2, \mathbf{p}^7, \mathbf{p}^9, \mathbf{p}^{10}\}$ *and* $\sum_{\mathbf{p}^l \in \bar{Y}^{\mathcal{N}_2}} \pi_l = 0.3 < \hat{\alpha}$. *We compute* $\kappa_6 = 2, \kappa_4 = 0, \kappa_5 = 1, \kappa_3 = 1, \kappa_7 = 2$ *and* $\kappa_2 = 1$. *Therefore,*

$$\hat{\mathcal{N}}_2 = \{C_6^1, C_6^2, C_5^1, C_3^1, C_7^1, C_7^2, C_2^1\}$$

and $\bar{Y}^{\hat{\mathcal{N}}_2} = \{\mathbf{p}^2, \mathbf{p}^5, \mathbf{p}^7, \mathbf{p}^8, \mathbf{p}^9, \mathbf{p}^{10}\}$. *We also compute*

$$\mathbf{s}^{Y^{\mathcal{N}_2}} = (0, 0, 3, 5, 0, 9, 9, 14, 19, 22) \rightarrow \text{UB}^{\mathcal{N}_2} = 22 \text{ and}$$
$$\mathbf{s}^{Y^{\hat{\mathcal{N}}_2}} = (0, 0, 3, 5, 0, 9, 9, 13, 17, 20) \rightarrow \text{LB}^{\mathcal{N}_2} = 20.$$

4.3.1.3 Branching scheme 2

Similarly to branching scheme 1, branching scheme 2 also branches over casets of realizations. Each node is associated with a set of excluded casets. Without loss of generality, we use the same notation $\mathcal{N}_u$ to represent the uth node in the tree. Branching scheme 2 differs from branching scheme 1 in two major ways. Firstly, in each node of branching scheme 2, a set of target casets can be excluded (note that this set can be an empty set) instead of one single target caset. This set of target casets is denoted by $\Omega(\mathcal{N}_u)$. Secondly, each level of the tree is associated with a certain activity.

The branching starts with the root node. The root node is branched into a number of child nodes associated with the first activity in the AL, each corresponding with the exclusion of a certain set of casets (remember that this set can be an empty set). Let us assume that activity i is the first activity in the AL. The first child node is associated with the exclusion of the set of casets $\{C_i^1, ..., C_i^{\zeta_i}\}$, the second child node is associated with the exclusion of the set of casets $\{C_i^1, ..., C_i^{\zeta_i - 1}\}$, and so on. Finally, the last node is associated with the exclusion of no caset. Each of these child nodes is then branched into its own children, which are associated with the second activity in the AL, and so on. Backtracking happens in a node if all its children have already been visited, if its set of eligible casets is an empty set or if the exclusion of its target casets is infeasible.

For branching scheme 2, $\text{LB}^{\mathcal{N}_u}$ is computed based on the following steps:

1. We construct $\hat{\mathcal{N}}_u$ which represents an extremely pessimistic case. $\hat{\mathcal{N}}_u$ initially contains all casets in $\mathcal{N}_u$.

2. Let activity i be the activity associated with the current node. For each activity $i' \in N$ that is positioned after activity i in the AL, add all casets $C_{i'}^{k_1}$ with $k_1 = 1, ..., \kappa_{i'}$ to $\hat{\mathcal{N}}_u$ where $\kappa_{i'}$ is the largest integer that satisfies the following condition:

$$\sum_{\mathbf{p}_l \in \bar{Y}^{\mathcal{N}_u} \cup \{C_{i'}^1, ..., C_{i'}^{\kappa_{i'}}\}} \pi_l \leq \hat{\alpha}.$$

3. $\mathrm{LB}^{\mathcal{N}_u} = s_{n+1}^{Y^{\hat{\mathcal{N}}_u}}$.

Similarly to branching scheme 1, every node $\mathcal{N}_u$ in our B&B tree is dominated if $\mathrm{LB}^{\mathcal{N}_u} \geq \mathrm{UB}^*$. Beware that this dominance rule is also considered in the tree that is presented in Figure 4.2.

Example 4.7. *Figure 4.2 depicts the B&B tree where branching scheme 2 is used in a best-first mode. Each node is represented by a square. The root node is branched into three nodes: $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$. Among these three nodes, node $\mathcal{N}_3$ is branched first since its lower bound is smaller than that of the other two nodes. $\mathcal{N}_3$ is branched into two nodes: $\mathcal{N}_4$ and $\mathcal{N}_5$. Node $\mathcal{N}_4$ whose lower bound is larger than the best upper bound (UB^*) found so far is eliminated from the three, whereas $\mathcal{N}_5$ is branched into its children ($\mathcal{N}_6$, $\mathcal{N}_7$ and $\mathcal{N}_8$). The next node to be branched is $\mathcal{N}_1$ because its lower bound is smaller than the lower bounds of all other unbranched nodes. The branching continues with the same logic until no unbranched node with a lower bound smaller than UB^* exists in the tree.*

All white nodes with a line over them are those that are left unbranched in the tree after the branching stopped. All gray nodes are those for which $\sum_{\mathbf{p}_l \in \bar{Y}^{\mathcal{N}}} \pi_l < \hat{\alpha}$ and $\mathrm{LB}^{\mathcal{N}} < \mathrm{UB}^* \leq \mathrm{UB}^{\mathcal{N}}$ and thus they are branched from and not eliminated. All green nodes are those for which $\sum_{\mathbf{p}_l \in \bar{Y}^{\mathcal{N}}} \pi_l \leq \hat{\alpha}$ and $\mathrm{UB}^{\mathcal{N}} < \mathrm{UB}^*$. All red nodes with a line over them are those for which $\sum_{\mathbf{p}_l \in \bar{Y}^{\mathcal{N}}} \pi_l > \hat{\alpha}$ or $\mathrm{LB}^{\mathcal{N}} \geq \mathrm{UB}^*$ and hence they are eliminated.

4.3.2 Improvements by hashing and listing

Although $\mathrm{LB}^{\mathcal{N}_u}$ is a valid lower bound and thus can be used to prune our B&B tree, its computation can be costly since it requires calling the optimization oracle $\mathcal{O}(\cdot)$. Therefore, one might be interested in finding a

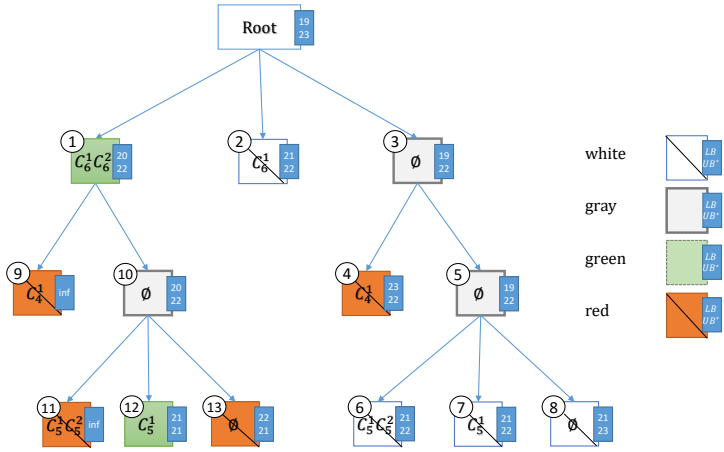


Figure 4.2: Branching scheme 2.

computationally much cheaper lower bounding approach. Assume $\mathbf{p}^{\hat{Y}}$ is a vector of durations for which $\mathcal{O}(\mathbf{p}^{\hat{Y}})$ has already been solved and $\mathbf{s}^{\hat{Y}}$ is its resulting optimal solution. If $\mathbf{p}^{\hat{Y}} \leq \mathbf{p}^{Y^{\mathcal{N}_u}}$, then $s^{\hat{Y}}_{n+1} \leq \mathrm{LB}^{\mathcal{N}_u}$. In our B&B algorithm, we call the optimization oracle $\mathcal{O}(\mathbf{p}^{Y^{\mathcal{N}_u}})$ only if for each $\mathbf{p}^{\hat{Y}} \leq \mathbf{p}^{Y^{\mathcal{N}_u}}$, the inequality $s^{\hat{Y}}_{n+1} < \mathrm{UB}^*$ is satisfied. Otherwise, the node $\mathcal{N}_u$ is dominated.

In our algorithm, every time the oracle $\mathcal{O}(\mathbf{p}^Y)$ is solved, we store the pair $(\mathbf{p}^Y, \mathbf{s}^Y)$ both in a hash table and in a linked list. The hash table is used to avoid calling $\mathcal{O}(\mathbf{p}^Y)$ more than once for the nodes with a common set of excluded realizations whereas the linked list is used for the lower bound computation. Before calling the optimization oracle to compute the lower bound, we check all pairs $(\mathbf{p}^Y, \mathbf{s}^Y)$ in the linked list and ensure that no pair in the linked list is sufficient to conclude the domination of the node. The members of the linked list are constantly ordered by the number of times they successfully caused a domination.

4.4 Computational results

In this section, we report computational results for our B&B algorithm. We also compare the performance of our B&B algorithm with the given MILP formulation in Section 4.2 and the branch-and-cut algorithm proposed by Lamas and Demeulemeester (2016). To implement the B&B algorithm, the MILP formulation and the B&C algorithm, Visual C++ 2010 and Cplex 12.5.1 were used. All computational results were obtained on a computer with Intel(R) Xeon(R) CPU E5-2699 v3 2.30 GHz (2 processors, 36 cores), 256GB of RAM and running under Windows Server 2012 R2. It is worth mentioning that in our experiments, at each time instant, 32 problem instances ran in parallel such that each instance was using only one thread (core). The remaining four cores were deliberately kept idle to deal with any possible overhead tasks and thus refrain such tasks from significantly influencing the results in our experiments.

We chose a memory limit of 10 GB and a time limit of one hour to solve each instance of the problem using any of the methods. The B&B method and the B&C method usually required about a few hundreds MB of RAM (note that the required memory to solve the B&C algorithm is significantly larger than that of our B&B algorithm). It is worth mentioning that, in our experiments, these two algorithms never exceeded the memory limit. The MILP formulation, on the other hand, requires a larger amount of memory which is often still less than 10 GB. This method exceeded the memory limit only once in our experiments. If an instance is solved within the time and memory limits, it is labeled *solved*, and otherwise *unsolved*.

4.4.1 Instance generation

All methods are tested on a set of instances that are composed of the PSPLIB instances. Only instances with 30 non-dummy activities are considered in this experiment. PSPLIB is a class of instances for the deterministic RCPSP (Kolisch and Sprecher, 1997), thus they need to be modified to suit our problem. The following modifications are applied on this set of instances: the activity durations $\tilde{p}_i$ for each non-dummy activity i follow a discretized beta distribution with shape parameters 2 and 5 that is mapped over the interval $[0.75\hat{p}_i, 1.625\hat{p}_i]$ where $\hat{p}_i$ is the duration of activity i that is given in the original instance. In order to reduce the number of experiments, we only consider the instances from the set J30 of PSPLIB with the following filename syntax: `J30X_1` ($\texttt{X} = 1, ..., 48$). The random generator's seed for the instance obtained from `J30X_1` equals `X`.

The size of the set $\mathfrak{P}$ can be extremely large. Any algorithm (including MILP solvers) that solves the problem might not be computationally tractable if $\mathfrak{P}$ is large. Therefore, we apply a sample average approximation technique to deal with the problem. We generate several sets of realizations with different sizes ($m = 100, 200, 400, 800$ or 1600) as explained in Lamas and Demeulemeester (2016). We select $1 - \hat{\alpha}$ from the set $\{0.99, 0.95, 0.90, 0.80\}$. For each combination of $(\mathtt{X}, m, (1 - \hat{\alpha}))$, an instance results and thus the total number of instances is $48 \times 5 \times 4 = 960$.

4.4.2 Overall results

We run our B&B algorithm on the set of instances described in Section 4.4.1 using different branching schemes and different priority rules. We report the overall results in Table 4.5. In separate experiments, we evaluate the performance of our B&B algorithm for each combination of a branching scheme and a priority rule. Branching schemes that are used in these experiments are branching scheme 1 (BS1) in depth-first mode, BS1 in a best-first mode and branching scheme 2 (BS2) in a best-first mode. Notice that we deliberately decide not to include BS2 in a depth-first mode in our experiments since the order in which the sets of excluded casets are evaluated in BS2 in a depth-first mode highly resembles that in BS1 in a depth-first mode. Also, preliminary results indicate that the required CPU time, the number of nodes and the number of oracle calls for these two approaches are very close. We use all priority rules described in Section 4.3.1.1 in these experiments.

Among all branching schemes, branching scheme 1 in a depth-first mode clearly performs the best. This better performance can be justified by expressing the importance of tight upper bounds. Obviously, a tight upper bound in the early stages of searching the tree can help pruning the low-quality branches. In a best-depth mode, such a tight upper bound is often obtained very late.

Our B&B algorithm performs best when Rule 3 is used to construct the activity list. This suggests that the total slack is the most important criterion, the number of casets is the second important criterion and the influence factor is the least important criterion. It also suggests that the choice of priority rule significantly influences the performance of our B&B algorithm.

The comparison between different settings in Table 4.5 is not very clear because the number of solved instances varies for different settings. In order to provide a better comparison between settings, each pair must

Priority rule	BS1		BS2
	Depth-first	Best-first	Best-first
Rule 1	339.60 (897)	408.98 (884)	422.01 (878)
Rule 2	347.49 (895)	394.20 (885)	389.61 (883)
Rule 3	**178.81 (925)**	213.08 (919)	207.64 (920)
Rule 4	211.58 (917)	233.99 (914)	238.86 (910)
Rule 5	374.06 (887)	447.94 (874)	469.47 (866)
Rule 6	495.57 (861)	624.88 (831)	656.77 (820)
Rule 7	540.97 (855)	611.26 (838)	620.38 (831)
Rule 8	536.24 (856)	608.88 (838)	615.14 (836)

Table 4.5: Average CPU times (in seconds) and number of solved instances within the time limit (out of 960) for different choices of priority rules and different branching schemes.

be separately compared in more detail. In such a comparison, one should only consider the instances that are solved to optimality in the settings under question. In Table 4.6, we compare two different settings, namely the combination of BS1 and Rule 1 (also denoted by the pair (BS1, Rule 1)) in a depth-first mode and (BS1, Rule 3) also in a depth-first mode. Let $\mathrm{CPU}_I(\mathrm{BS1}, \mathrm{Rule\ 1})$ be the CPU time required to optimally solve instance I using the former setting and $\mathrm{CPU}_I(\mathrm{BS1}, \mathrm{Rule\ 3})$ be the CPU time required to optimally solve instance I using the latter setting. We compute the average percentage deviation of the required CPU time using (BS1, Rule 3) in a depth-first mode from the required CPU time using (BS1, Rule 1) in a depth-first mode as follows:

$$\mathrm{avg}\left\{\frac{\mathrm{CPU}_I(\mathrm{BS1}, \mathrm{Rule\ 3}) - \mathrm{CPU}_I(\mathrm{BS1}, \mathrm{Rule\ 1})}{\mathrm{CPU}_I(\mathrm{BS1}, \mathrm{Rule\ 1})} \times 100\%\right\}.$$

According to the results that are presented in Table 4.6, for every choice of $1-\hat{\alpha}$ and m, (BS1, Rule 3) performs better than (BS1, Rule 1) (negative average percent deviation). We notice that by increasing the number of realizations (m) and also by decreasing the confidence level ($1-\hat{\alpha}$), the associated average percent deviation decreases. In other words, by increasing the number of realizations and also by decreasing the confidence level, the difference between the performances of (BS1, Rule 3) and (BS1, Rule 1) becomes more significant. For instance, when $1-\hat{\alpha} = 0.95$ and $m = 200$, (BS1, Rule 3) performs about two times faster than (BS1, Rule 1) whereas

$1-\hat{\alpha}$	m				
	100	200	400	800	1600
0.99	-8.91 (48)	-7.21 (48)	-39.37 (48)	-32.65 (48)	-47.94 (48)
0.95	-42.27 (48)	-53.72(47)	-61.07 (47)	-65.85 (46)	-78.26 (44)
0.90	-48.64 (46)	-70.54 (47)	-71.63 (46)	-65.44 (44)	-80.60 (43)
0.80	-69.19 (45)	-83.97 (45)	-74.74 (40)	-85.25 (37)	-94.18 (30)

Table 4.6: The average percentage deviation of the required CPU time using (BS1, Rule 3) in a depth-first mode from the required CPU time using (BS1, Rule 1) in a depth-first mode (in percentage) and the number of instances solved in both settings (out of 48) for different choices of $1-\hat{\alpha}$ and m.

when $1-\hat{\alpha}=0.80$ and $m=1600$, (BS1, Rule 3) performs about 17 times faster than (BS1, Rule 1).

Among all combinations, (BS1, Rule 3) in a depth-first mode performs the best. In the remainder of this chapter, we only report the results of our B&B for this combination and the results associated with other combinations are ignored. Note that in the following subsections, for the sake of simplicity, we mention no setting and instead we simply use the notion 'our B&B'.

4.4.3 Detailed results

In Table 4.7, for each pair $((1-\hat{\alpha}),m)$, we report the number of instances that are solved within the time limit (Solved), the average and maximum CPU times (avg(CPU) and max(CPU)), the average number of casets (avg($|\mathbf{C}^E|$)), the average number of nodes visited in the tree (avg(NN)) and the average number of times the optimization oracle is called (avg(OC)).

We observe that by decreasing $1-\hat{\alpha}$ from 0.99 to 0.80 and/or by increasing the number of realizations (m), the average number of casets, the average CPU times, the average number of nodes and the average number of oracle calls are often increased whereas the average number of solved instances is decreased or remains unchanged.

We report that the average number of oracle calls is increased almost linearly by increasing the number of realizations, which is the main strength of this approach. However, the average number of oracle calls is increased exponentially with an increase in the number of casets (see Figure 4.3).

$1-\hat{\alpha}$	m	Solved	CPU		avg($\lvert\mathbf{C}^E\rvert$)	avg(NN)	avg(OC)
			avg	max			
0.99	100	48	2.71	118.56	6.94	6.73	7.73
	200	48	3.29	133.55	10.38	16.33	14.40
	400	48	18.95	859.73	14.46	45.38	28.83
	800	48	22.76	943.68	17.56	136.98	66.63
	1600	48	72.04	2994.27	21.10	243.94	84.71
0.95	100	48	32.61	1360.92	22.19	113.96	71.73
	200	48	60.56	2611.33	26.02	327.60	161.38
	400	47	95.85	3600.00	30.35	1076.27	380.25
	800	46	171.39	3600.00	33.98	3031.90	825.19
	1600	46	206.83	3600.00	37.19	5484.21	1290.60
0.90	100	47	101.44	3600.00	31.75	732.35	358.17
	200	47	110.79	3600.00	36.42	1893.04	729.15
	400	46	184.42	3600.00	40.46	5725.63	1586.52
	800	46	263.40	3600.00	44.27	18093.48	4176.90
	1600	45	296.78	3600.00	48.19	26253.33	5392.23
0.80	100	46	258.06	3600.00	44.42	6028.85	2270.23
	200	46	212.91	3600.00	49.08	17682.98	4882.92
	400	43	402.35	3600.00	53.58	39607.83	9144.67
	800	42	509.95	3600.00	57.79	136760.71	24427.54
	1600	42	549.09	3600.00	61.98	156366.33	27565.60

Table 4.7: The detailed computational results for our B&B algorithm.

4.4.3.1 Quality of the lower bound

One of the main features of our B&B algorithm is the lower bound computation. To show the strength of the proposed lower bound, we report the average deviation of the lower bound in the root node from the objective value of the best found (optimal) solution in Table 4.8. This average deviation is computed as follows:

$$\text{avg}\left\{\frac{\text{LB}^{\mathcal{N}_0} - UB^*}{UB^*} \times 100\%\right\}.$$

Notice that the computation of the lower bound in the root node is exactly the same for the two branching schemes. The results in Table 4.8 suggest that by decreasing the confidence level and by increasing the number of realizations, the quality of the lower bound is generally decreased.

4.4.3.2 Impacts of hashing and listing

In Section 4.3.2, we discussed two improvement techniques, namely exploiting a hash table (HT) and a linked list (LL). Table 4.9 demonstrates

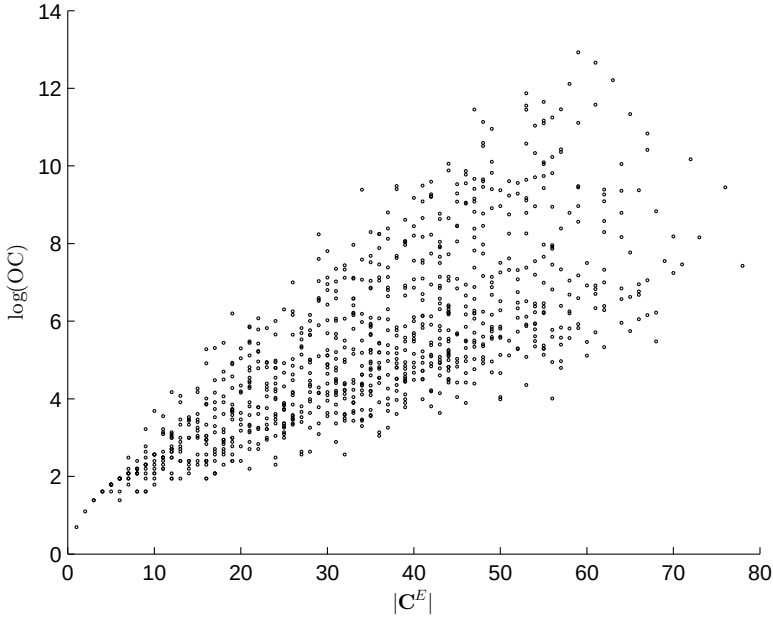


Figure 4.3: Logarithm of number of oracle calls vs. number of casets.

$1-\hat{\alpha}$	m				
	100	200	400	800	1600
0.99	-2.23	-2.63	-3.16	-3.26	-3.17
0.95	-5.86	-5.71	-6.28	-6.12	-6.35
0.90	-7.68	-8.39	-8.15	-8.25	-8.25
0.80	-10.69	-11.31	-11.06	-11.44	-11.67

Table 4.8: The average percent deviation between the lower bound and the objective value of the best found (or optimal) solution for different choices of $1-\hat{\alpha}$ and m.

their effects on the performance of our B&B algorithm. The first row represents the setting in which both the hash table and the linked list are exploited. The second and third row represent the settings in which either of the two improving techniques is not exploited. Finally, the last low represents the setting in which none of them is exploited. Based on these results, we notice that implementing the hash table slightly improves the

Setting	Solved (out of 960)	avg(CPU)	avg(OC)
B&B	925	178.81	4173.27
B&B - HT	925	180.81	4514.47
B&B - LL	913	239.61	19483.30
B&B - HT - LL	913	241.19	19686.32

Table 4.9: The effect of implementing the hash table (HT) and/or the linked list (LL) on our B&B algorithm.

$1-\hat{\alpha}$	m				
	3200	6400	12800	25600	51200
0.99	35.66 (48)	92.69 (48)	134.54 (47)	160.03 (47)	245.81 (45)
0.95	247.56 (45)	347.23 (44)	388.62 (45)	381.67 (44)	413.95 (44)
0.90	406.65 (44)	484.43 (44)	517.01 (42)	675.13 (41)	737.77 (40)
0.80	697.72 (40)	846.37 (39)	946.33 (38)	1029.67 (36)	1037.80 (35)

Table 4.10: The average CPU time and the number of instances solved (out of 48) for different choices of $1-\hat{\alpha}$ and large m values.

performance of our B&B algorithm whereas the improvement resulting from the implementation of the linked list is significant.

4.4.3.3 Results for instances with a large number of realizations

Readers may be interested in the performance of our B&B algorithm when larger sets of realizations are considered. In this part, we present the results of an additional set of instances. This set is generated similarly to the set of instances introduced in Section 4.4.1, except that $m = 3200, 6400, 12800, 25600$ or 51200. Table 4.10 reports the average CPU times and the numbers of instances solved for different choices of $1-\hat{\alpha}$ and these large m values. We notice that our B&B algorithm can solve 35 (out of 48) instances even for the most difficult setting, *i.e.*, the setting where $m = 51200$ and $1-\hat{\alpha} = 0.80$. The main reason of such a good performance lies in the way the casets are introduced. Having a fixed range of integer numbers, the number of casets remains almost constant by increasing the number of realizations. Note that the number of oracle calls and as such the required CPU times are exponentially increased only by increasing the number of casets, as it has been shown in Figure 4.3. Therefore, because the number of casets remains almost constant by increasing the number of realizations, we conclude that increasing the number of realizations does not lead to a significant increase in CPU times.

$1-\hat{\alpha}$	m	Solved	CPU		avg($\lvert\mathbf{C}^E\rvert$)	avg(NN)	avg(OC)
			avg	max			
0.99	100	48	3.99	166.937	13.15	12.52	13.50
	200	48	22.70	1007.54	20.77	45.50	41.33
	400	48	89.79	3490.83	29.19	238.63	184.52
	800	47	120.67	3600.00	36.29	1274.45	659.21
	1600	46	249.27	3600.00	42.42	6458.27	2583.79
0.95	100	47	109.14	3600.00	42.06	835.25	628.56
	200	45	313.76	3600.00	52.75	5612.54	2950.17
	400	44	480.87	3600.00	62.13	54531.02	19719.38
	800	39	842.35	3600.00	69.25	218721.46	50665.90
	1600	36	1178.17	3600.00	75.50	399935.52	85520.31
0.90	100	45	313.66	3600.00	61.23	12494.23	6862.81
	200	42	570.21	3600.00	73.31	74333.40	27812.06
	400	34	1274.74	3600.00	82.67	324515.38	86804.75
	800	33	1294.26	3600.00	91.17	441609.19	92741.42
	1600	28	1582.64	3600.00	97.98	576703.73	115854.06
0.80	100	38	903.09	3600.00	85.13	175219.60	67898.67
	200	34	1266.76	3600.00	97.88	330257.13	98129.88
	400	29	1591.19	3600.00	108.33	522442.42	126655.79
	800	27	1700.57	3600.00	116.96	653758.15	128094.58
	1600	24	1905.06	3600.00	124.79	935444.83	167067.04

Table 4.11: The detailed computational results for our B&B algorithm ran on instances with medium variances.

4.4.3.4 Results for instances with medium and high variances

The activity durations of the instances introduced in Section 4.4.1 are generated from the range $[0.75\hat{p}_i, 1.625\hat{p}_i]$ which represents a low variance. In most papers that are dealing with project scheduling where activity durations are stochastic, two wider ranges are also used: a range of $[0.5\hat{p}_i, 2.25\hat{p}_i]$ which represents a medium variance in activity durations and a range of $[0.25\hat{p}_i, 2.875\hat{p}_i]$ which represents a high variance in activity durations. We introduce two additional sets of instances that are generated similarly to the set of instances introduced in Section 4.4.1, except that the medium and the high variance ranges are used to generate the activity durations. Tables 4.11 and 4.12 show the detailed computational results for our B&B algorithm ran on instances with medium and high variances, respectively. As expected, by increasing the range (variance), the number of casets and as such the CPU times are increased whereas the number of solved instances is decreased. It is interesting to see that our B&B algorithm can also solve most of the instances with medium

$1-\hat{\alpha}$	m	Solved	CPU		avg($\lvert\mathbf{C}^E\rvert$)	avg(NN)	avg(OC)
			avg	max			
0.99	100	48	11.08	502.98	15.31	14.42	15.35
	200	48	29.25	1189.15	26.15	68.00	65.52
	400	47	141.76	3600.00	39.67	576.04	495.94
	800	45	319.38	3600.00	50.75	4691.56	2697.77
	1600	43	442.91	3600.00	62.40	26714.71	10894.48
0.95	100	47	131.68	3600.00	53.27	1199.21	965.15
	200	43	416.64	3600.00	70.67	12984.54	7192.79
	400	36	957.54	3600.00	87.29	162026.98	63291.98
	800	31	1404.69	3600.00	100.33	357048.48	104380.56
	1600	27	1632.23	3600.00	112.33	529853.10	126483.63
0.90	100	43	470.28	3600.00	81.10	20620.33	13079.69
	200	36	1053.60	3600.00	100.50	191257.79	82103.69
	400	27	1769.10	3600.00	117.21	427891.35	141403.94
	800	25	1880.22	3600.00	129.98	615474.25	158701.44
	1600	21	2274.00	3600.00	142.96	851616.31	183599.63
0.80	100	32	1408.39	3600.00	118.40	227325.42	109373.13
	200	22	2090.18	3600.00	138.65	462194.48	170729.17
	400	18	2440.46	3600.00	155.69	645457.58	195175.23
	800	16	2653.74	3600.00	169.06	910299.63	216549.46
	1600	13	2892.69	3600.00	181.88	1111028.08	232467.46

Table 4.12: The detailed computational result for our B&B algorithm ran on instances with high variances.

variance and a reasonably large number of instances with high variance to optimality within the time limit.

4.4.4 Comparison with other methods

We compare our B&B algorithm with the mathematical formulations proposed in Section 4.2 and the B&C algorithm proposed by Lamas and Demeulemeester (2016). For each pair $(1-\hat{\alpha}, m)$, Table 4.13 reports the number of solved instances within different time limits: 10 seconds (10s), 1 minute (1m), 10 minutes (10m) and 1 hour (1h). We observe that our B&B algorithm clearly outperforms the mathematical formulation and the B&C algorithm in all settings.

As we decrease $(1-\hat{\alpha})$, the number of solved instances is decreased for all three methods. The same behavior is noticed when we increase the number of realizations. Intriguingly we notice that within 10 seconds, our B&B algorithm can solve more instances than the number of instances that are solved within one hour by the mathematical formulation. Also within one minute, our B&B algorithm can solve more instances than

$1-\hat{\alpha}$	m	B&B				SAA-RCPSP-MILP2				B&C			
		10s	1m	10m	1h	10s	1m	10m	1h	10s	1m	10m	1h
0.99	100	47	47	48	48	26	30	33	38	39	44	47	47
	200	47	47	48	48	26	30	32	35	38	43	47	47
	400	46	47	47	48	24	29	32	36	38	40	46	46
	800	43	46	47	48	22	28	31	34	33	39	43	46
	1600	43	45	47	48	19	30	32	35	28	37	42	46
0.95	100	44	46	47	48	21	30	30	37	35	39	43	46
	200	42	45	47	48	19	30	31	36	30	39	42	44
	400	41	43	47	47	18	29	32	36	24	34	40	43
	800	42	43	45	46	9	24	32	37	16	27	38	40
	1600	41	43	44	46	6	25	29	32	6	22	31	39
0.90	100	41	44	46	47	17	28	31	37	28	38	42	44
	200	42	42	46	47	10	25	31	35	23	32	38	42
	400	40	43	45	46	8	23	31	33	12	26	38	40
	800	40	41	43	46	5	19	31	32	9	19	29	39
	1600	36	41	43	45	0	10	26	31	0	7	23	32
0.80	100	39	41	44	46	9	22	30	32	18	29	40	42
	200	33	41	44	46	7	22	30	31	10	25	35	39
	400	32	38	42	43	5	13	28	29	4	15	27	36
	800	30	35	40	42	1	5	23	28	0	8	21	29
	1600	31	33	40	42	0	4	10	24	0	0	13	19

Table 4.13: The number of instances solved to optimality for different time limits (10 seconds, 1 minute, 10 minutes and 1 hour), different methods and different choices of $1-\hat{\alpha}$ and large m values.

the number of instances that are solved within one hour using the B&C algorithm. More interestingly, in many very difficult settings (for example the setting where $(1-\hat{\alpha}) = 0.80$ and $m = 1600$), the number of instances that are solved within 10 seconds by our B&B algorithm is much more than the number of instances that are solved within one hour using the B&C algorithm.

As is also clear in the table, B&C is generally performing better than the MILP formulation. However, when $1-\hat{\alpha} \leq 0.90$ and $m \geq 400$ the MILP formulation sometimes performs better, specially for the smaller time limits. This somewhat unexpected result might be because of the pre-processing steps in the B&C algorithm.

4.5 Discussion: general CCP problem

Although our proposed B&B algorithm is applied to solve the SAA-RCPSP, it can be used to solve any CCP problem with discrete random rhs vector provided that an optimal solution methodology exists for its deterministic counterpart. For instance, consider the classical trans-

portation problem with random discrete demand vector. One could use the same B&B algorithm proposed in Section 4.3 to solve the chance-constrained version of the classical transportation problem. In this case, $\mathcal{O}(.)$ should be an oracle that optimally solves the deterministic transportation problem.

4.6 Summary and conclusion

In this chapter, we propose a novel B&B algorithm that solves the SAA-RCPSP in a much more efficient manner than the methods that are already existing in the literature. The goal in SAA-RCPSP is to select a subset of realizations, for which the optimal solution must be feasible, such that the resulting confidence level is at least $(1 - \hat{\alpha})$. Instead of branching over realizations, we branch over casets of realizations and thus the complexity of the method is a function of the number of casets rather than a function of the number of realizations. If the activity realizations are discrete, then the number of casets usually is increased only slightly by increasing the number of the realizations.

In our experiments, we tested different priority rules for activities, based on which the B&B tree is constructed. We noticed that, among several sorting criteria, sorting activities based on smaller total slack times significantly improves the performance of our B&B algorithm. The performance analyses show that the efficiency of the B&B algorithm is mainly due to the concepts of casets, since the number of casets increases moderately as we increase the number of realizations. The latter property, however, holds if the number of realizations is significantly larger than the numbers of activity modes.

We ran our B&B algorithm together with a MILP formulation and a B&C algorithm on benchmark instances with a size of 30 activities. The B&B algorithm outperforms the MILP formulation and the B&C algorithm in terms of computational times as well as in terms of the number of solved instances within the time limit.

Part II

A generic single machine scheduling problem

Chapter 5

Introducing GSMSP: a single-machine scheduling problem with time windows and precedence constraints

The beauty of mathematics only shows itself to more patient followers.

- Maryam Mirzakhani

Scheduling problems arise in production planning (Sule, 2007), in balancing processes (Shirazi et al., 1995), in telecommunication (Nemeth et al., 1997) and more generally in all situations in which scarce resources are to be allocated to jobs over time (Pinedo, 2008). Depending on the application, the corresponding scheduling problem can be such that each job must be processed within a given time window, where the lower bound (*release date* or *ready time*) of this time window represents the earliest start of the execution of the job and the upper bound (*deadline*) corresponds with the latest acceptable completion time, for instance the ultimate delivery time agreed upon with the customer (Gordon et al., 1997;

This chapter together with Chapters 6 and 7 constitute our paper that is published in *Journal of Scheduling* (Davari et al., 2016).

Pan and Shi, 2005; Xu and Parnas, 1990). For some of these applications, only release dates or only deadlines are considered (Jouglet et al., 2004; Pan, 2003; Posner, 1985; Tanaka and Fujikuma, 2012). In practice, a job often also needs to be processed before or after other jobs, *e.g.*, due to tool or fixture restrictions or for other case-dependent technological reasons, which leads to *precedence constraints* (Lawler, 1978; Potts and Van Wassenhove, 1985; Tanaka and Sato, 2013). Finally, the contract with a client can also contain clauses that stipulate that penalties must be paid when the execution of a job is not completed before a reference date (*due date*) (Abdul-Razaq and Potts, 1988; Dyer and Wolsey, 1990; Ibaraki and Nakamura, 1994; Jouglet et al., 2004; Talla Nobibon and Leus, 2011; Tanaka and Fujikuma, 2012).

In this chapter, we study a generic single-machine scheduling problem (which is referred to as GSMSP) with total weighted tardiness (TWT) penalties. In the standard three-field notation introduced by Graham et al. (1979), GSMSP can be denoted as $1|r_j, \delta_j, prec|\sum w_j T_j$: the execution of each job is constrained to take place within a time window, and we assume the corresponding deadline to be greater than or equal to a due date, which is the reference for computing the tardiness of the job. The scheduling decisions are also subject to precedence constraints.

The remainder of this chapter is structured as follows. In Section 5.1 we briefly summarize the state of the art. In Section 5.2 we provide some definitions and a formal problem statement and in Section 5.3 we propose two different integer programming formulations. In Section 5.4 we explain the instance generation. Section 5.5 presents computational results and Section 5.6 is devoted to summary and conclusions.

5.1 Literature review

Abdul-Razaq et al. (1990) survey different branch-and-bound (B&B) algorithms for $1||\sum w_j T_j$. A benchmark algorithm is the B&B procedure of Potts (1985); an older reference is Held and Karp (1962), who present a dynamic programming (DP) approach. Abdul-Razaq and Potts (1988) introduce a DP-based approach to obtain tight lower bounds for the generalized version of the problem where the cost function is piecewise linear. They examine their lower bounds in a B&B algorithm and solve small instances (with at most 25 jobs) to optimality. Ibaraki and Nakamura (1994) extend their work and construct an exact method, called Successive Sublimation Dynamic Programming (SSDP), which solves medium-sized

instances (with up to 50 jobs). Tanaka et al. (2009) improve the SSDP of Ibaraki and Nakamura (1994) and succeed in solving reasonably large instances (with up to 300 jobs) of $1||\sum w_j T_j$ within acceptable runtimes.

Single-machine scheduling for TWT with (possibly unequal) release dates ($1|r_j|\sum w_j T_j$) has also been studied by several authors. Akturk and Ozdemir (2000, 2001) and Jouglet et al. (2004) develop B&B algorithms that solve small instances. Van den Akker et al. (2010) propose a time-indexed formulation and a method based on column generation to solve this problem with identical processing times. Tanaka and Fujikuma (2012) present an SSDP algorithm that can solve instances of $1|r_j|\sum w_j T_j$ with up to 100 jobs.

There are only few papers dealing with single-machine scheduling with deadlines and/or precedence constraints. Among these, we cite Posner (1985) and Pan (2003), who propose B&B algorithms for $1|\delta_j|\sum w_j C_j$, Pan and Shi (2005), who develop a B&B algorithm to solve $1|r_j, \delta_j|\sum w_j C_j$, Lawler (1978) and Potts and Van Wassenhove (1985), who present B&B algorithms to solve $1|prec|\sum w_j C_j$, and Tang et al. (2007), who propose a hybrid backward and forward dynamic-programming-based Lagrangian relaxation to compute upper and lower bounds for $1|prec|\sum w_j T_j$. Tanaka and Sato (2013) also propose an SSDP algorithm to solve a generalization of $1|prec|\sum w_j T_j$ (piecewise linear cost function). To the best of our knowledge, scheduling problems with release dates, deadlines and precedence constraints have not yet been studied in the literature. The goal of this chapter and the following two chapters is to fill this gap and to propose efficient B&B algorithms that solve all the foregoing subproblems within limited computation times.

5.2 Problem description

The jobs to be scheduled are gathered in set $N = \{1, 2, \ldots, n\}$. Job i is characterized by a processing time p_i, a release date r_i, a due date d_i, a deadline δ_i, and a weight w_i, which represents the cost per unit time of delay beyond d_i. Jobs can neither be processed before their release dates nor after their deadlines ($0 \leq r_i \leq \delta_i$). Precedence constraints are represented by a graph $G = (N', A)$, where $N' = N \cup \{0, n+1\}$, with 0 a dummy start job and $n+1$ a dummy end. Each arc $(i, j) \in A$ implies that job i must be executed before job j (job i is a predecessor of job j). We will assume that $G(N', A)$ is its own transitive reduction, that is, no transitive arcs are included in A. Let $\mathcal{P}_i$ be the set of all predecessors

of job i in A ($\mathcal{P}_i = \{k|(k,i) \in A\}$) and $\mathcal{Q}_j$ the set of successors of job i ($\mathcal{Q}_i = \{k|(i,k) \in A\}$). We also define an associated graph $\hat{G} = (N', \hat{A})$ as the transitive closure of G. We assume that $\mathcal{P}_0 = \mathcal{Q}_{n+1} = \emptyset$, and that all jobs are successor of 0 and predecessor of $n+1$ in $\hat{G}$ (apart from the jobs themselves).

Throughout this chapter and also the next two chapters, we use the term 'sequencing' to refer to ordering the jobs (establishing a permutation) whereas 'scheduling' means that start (or end) times are determined. We denote by π an arbitrary sequence of jobs, where π_k represents the job at the k^{th} position in that sequence. Let $\pi^{-1}(i)$ be the position of job i in π; we only consider sequences for which $\pi^{-1}(i) < \pi^{-1}(j)$ for all $(i,j) \in A$. Value C_i is the completion time of job i. Each sequence π implies a schedule, as follows:

$$C_{\pi_i} = \begin{cases} \max\{r_{\pi_i}, C_{\pi_{i-1}}\} + p_{\pi_i} & \text{if } i > 1 \\ r_{\pi_i} + p_{\pi_i} & \text{if } i = 1 \end{cases}.$$

Equivalently, the end of job i according to sequence π can also be written as $C_i(\pi)$. We denote by $\mathcal{D}$ the set of all feasible permutations, where a permutation π is feasible ($\pi \in \mathcal{D}$) if and only if it generates a feasible schedule, which means that

$$r_{\pi_i} + p_{\pi_i} \leq C_{\pi_i} \leq \delta_{\pi_i} \qquad \forall i \in N.$$

Note that the set $\mathcal{D}$ may be empty.

The weighted tardiness associated with the job at the i^{th} position in the sequence π is given by $W(\pi_i) = w_{\pi_i} (C_{\pi_i} - d_{\pi_i})^+$, where $x^+ = \max\{0, x\}$. A conceptual formulation of the problem GSMSP studied in this chapter is given in the following:

$$\text{GSMSP}: \min_{\pi \in \mathcal{D}} \text{TWT}(\pi) = \sum_{i=1}^{n} W(\pi_i). \tag{5.1}$$

This problem is at least as hard as $1||\sum w_i T_i$, which is known to be strongly NP-hard (Lawler, 1977; Lenstra et al., 1977; Pinedo, 2008). A stronger result is that the mere verification of the existence of a feasible schedule that respects a set of ready times and deadlines is already NP-complete (Garey and Johnson, 1979, problem SS1, page 236); we do not, however, incorporate the feasibility check as a formal part of the problem statement.

Job i	p_i	r_i	d_i	δ_i	w_i
Job 1	2	3	10	14	1
Job 2	3	4	11	13	2
Job 3	4	3	8	15	3
Job 4	2	2	6	9	1

Table 5.1: Job characteristics.

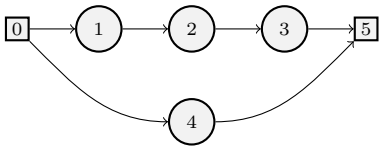


Figure 5.1: Precedence graph $G(N', A)$.

Example. *Consider the following instance of* GSMSP *with* $n = 4$ *jobs. The processing times, release dates, due dates, deadlines and weights of the jobs are given in Table 5.1. The graph representing the precedence constraints is depicted in Figure 5.1, with arc set* $A = \{(0,1), (0,4), (1,2), (2,3), (3,5), (4,5)\}$.

An optimal solution to this instance is $\pi = (4,1,2,3)$*, which leads to the schedule* $C_1 = 6$*,* $C_2 = 9$*,* $C_3 = 13$ *and* $C_4 = 4$*. The objective value is* $w_4 \times 0 + w_1 \times 0 + w_2 \times 0 + w_3 \times (13 - 8) = 3 \times 5 = 15$.

5.3 Mathematical formulations

The conceptual formulation for GSMSP presented in the previous section is not linear, therefore it cannot be used by a standard (linear) mixed-integer programming (MIP) solver. In this section, we propose an Assignment Formulation (ASF) and a Time-Indexed Formulation (TIF) for the problem. These formulations are adaptations of those presented in Keha et al. (2009) and Talla Nobibon and Leus (2011).

5.3.1 Assignment formulation

We use binary decision variables $x_{is} \in \{0,1\}$ $(i \in N, s \in \{1,2,\ldots,n\})$, which identify the position of jobs in the sequence so that x_{is} is equal to 1 if job i is the s^{th} job processed and equal to 0 otherwise. In other

words, $x_{is} = 1$ if and only if $\pi_s = i$. We also use additional continuous variables $T_i \geq 0$ representing the tardiness of job $i \in N$ and continuous variables $\tau_s \geq 0$ representing the machine idle time immediately before the execution of the s^{th} job. The MIP formulation is given by

$$\text{ASF}: \quad \min \quad \sum_{i=1}^{n} w_i T_i \tag{5.2}$$

subject to

$$\sum_{s=1}^{n} x_{is} = 1 \quad \forall i \in N \tag{5.3}$$

$$\sum_{i=1}^{n} x_{is} = 1 \quad \forall s \in \{1, 2, \ldots, n\} \tag{5.4}$$

$$\sum_{s=1}^{n} x_{is} s \leq \sum_{t=1}^{n} x_{jt} t - 1 \quad \forall (i,j) \in A \tag{5.5}$$

$$\tau_s \geq \sum_{i=1}^{n} x_{is} r_i - \sum_{t=1}^{s-1} \left(\sum_{i=1}^{n} (x_{it} p_i) + \tau_t \right) \quad \forall s \in N \tag{5.6}$$

$$\sum_{t=1}^{s} \tau_t + \sum_{t=1}^{s-1} \sum_{i=1}^{n} p_i x_{it} + \sum_{i=1}^{n} ((p_i - \delta_i) x_{is}) \leq 0 \quad \forall s \in N \tag{5.7}$$

$$T_i \geq \sum_{t=1}^{s} \tau_t + \sum_{t=1}^{s-1} \sum_{j=1}^{n} p_j x_{jt} + p_i - d_i - (1 - x_{is}) M_i \quad \forall i \in N, s \in N \tag{5.8}$$

$$x_{is} \in \{0,1\}, \tau_s, T_i \geq 0 \quad \forall i \in N, s \in \{1, 2, \ldots, n\}. \tag{5.9}$$

The objective function (5.2) is a reformulation of (5.1). The set of constraints (5.3) ensures that all jobs are executed. Constraints (5.4) check that each position in the sequence is occupied by exactly one job. The set of constraints (5.5) enforces the precedence restrictions. The set of equations (5.6) computes the idle time of the machine between the jobs in positions $s-1$ and s, and ensures that each job is not started before its release date. In this set of constraints, $\sum_{t=1}^{s-1} (\sum_{i=1}^{n} (x_{it} p_i) + \tau_t)$ equals the completion time of the $(s-1)^{\text{th}}$ job. Constraints (5.7) ensure that each job is not completed after its deadline, where $\sum_{t=1}^{s} \tau_t + \sum_{t=1}^{s-1} \sum_{i=1}^{n} p_i x_{it}$ is the start time of the s^{th} job. Constraints (5.8) compute the correct

value of the tardiness of job i, with $M_i = \delta_i - d_i$ the maximum tardiness of job i.

A variant of ASF is obtained by replacing the set of constraints (5.5) by the following:

$$\sum_{s=v}^{n} x_{is} + \sum_{s=1}^{v} x_{js} \leq 1 \quad \forall (i,j) \in A, \forall v \in \{1, \ldots, n\}. \tag{5.10}$$

We refer to this alternative formulation as ASF′. We have the following result:

Lemma 5.1. ASF′ *is stronger than* ASF.

Proof. Consider the set of constraints (5.10). For each $(i,j) \in A$, the following inequalities hold:

$$\begin{aligned}
&x_{i1} + \cdots + x_{in} \leq 1 - x_{j1} = x_{j2} + \cdots + x_{jn} \\
&x_{i2} + \cdots + x_{in} \leq 1 - x_{j1} - x_{j2} = x_{j3} + \cdots + x_{jn} \\
&\qquad\qquad\qquad\qquad \vdots \\
&x_{i(n-1)} + x_{in} \leq 1 - x_{j1} - \cdots - x_{j(n-1)} = x_{jn} \\
&x_{in} \leq 1 - x_{j1} - \cdots - x_{jn} = x_{j1} + \cdots + x_{jn} - 1
\end{aligned}$$

By adding the above inequalities, we obtain

$$\begin{aligned}
x_{i1} + 2x_{i2} + 3x_{i3} + \cdots + nx_{in} \leq \\
x_{j1} + 2x_{j2} + 3x_{j3} + \cdots + nx_{jn} - 1.
\end{aligned}$$

This is exactly the associated constraint in the set of constraints (5.5). As a result, the solution space of the LP relaxation of ASF′ is included in that of ASF. To show that the inclusion is strict, consider the following fractional values for the decision variables corresponding with a couple $(i,j) \in A$: $x_{i1} = x_{i5} = 0.5$ and $x_{j4} = 1$. These values can be seen to respect the weak but not the strong formulation. □

The number of constraints in (5.10) is much higher than in (5.5). As a result, the additional computational effort needed to process this higher number of constraints might offset the improvement of a stronger bound, and we will empirically compare the performance of the two variants in Section 5.5.

5.3.2 Time-indexed formulation

Let T_S (respectively T_E) be a lower (respectively upper) bound on the time the execution of any job can be completed; we compute these values as $T_S = \min\{r_i + p_i | i \in N\}$ and $T_E = \max\{\delta_i | i \in N\}$. The time-indexed formulation uses binary decision variables $x_{it} \in \{0, 1\}$, for $i \in N$ and $T_S \leq t \leq T_E$, where $x_{it} = 1$ if job i is completed (exactly) at time t and $x_{it} = 0$ otherwise. We also introduce the set of parameters $T_{it} = (t - d_i)^+$, representing the tardiness of job i when it finishes at time t. The time-indexed formulation is given by

$$\text{TIF}: \quad \min \quad \sum_{i=1}^{n} \sum_{t=r_i+p_i}^{\delta_i} w_i T_{it} x_{it} \tag{5.11}$$

subject to

$$\sum_{i=1}^{n} \sum_{s=\max\{t, r_i+p_i\}}^{\min\{\delta_i, t+p_i-1\}} x_{is} \leq 1 \qquad \forall t, T_S \leq t \leq T_E \tag{5.12}$$

$$\sum_{t=r_i+p_i}^{\delta_i} x_{it} = 1 \qquad \forall i \in N \tag{5.13}$$

$$\sum_{s=r_i+p_i}^{\delta_i} x_{is} s \leq \sum_{t=r_j+p_j}^{\delta_j} x_{jt} t - p_j \quad \forall (i, j) \in A \tag{5.14}$$

$$x_{it} \in \{0, 1\} \quad i \in N,\ r_i + p_i \leq t \leq \delta_i. \tag{5.15}$$

The set of constraints (5.12) eliminates the parts of the solution space where the jobs overlap. The constraint set (5.13) ensures that all jobs are scheduled exactly once. We enforce precedence constraints in the formulation using the set of constraints (5.14).

Similarly as for the assignment formulation, we introduce an alternative formulation TIF′ by replacing the set of constraints (5.14) by the following:

$$\sum_{s=t}^{\delta_i} x_{is} + \sum_{s=r_j+p_j}^{t-p_i} x_{js} \leq 1 \tag{5.16}$$

$$\forall (i, j) \in A; \forall t, \max\{r_i, r_j + p_j\} + p_i \leq t \leq \min\{\delta_i, \delta_j + p_i\}.$$

Lemma 5.2. (From Artigues et al., 2008; Christofides et al., 1987) TIF′ is stronger than TIF.

As explained for the assignment formulation, the performance of the new formulation is not necessarily better. In fact, it can be much worse than TIF, since in a time-indexed formulation the number of additional constraints is quite large (pseudo-polynomial).

5.4 Instance generation

To the best of our knowledge, there are no benchmark sets of instances of problem GSMSP available, and so we have generated our own set of instances, which is referred to as Ins. The set Ins consists of the two disjoint subsets Ins^S and Ins^L): Ins^S contains instances with small processing times and Ins^L holds instances with large processing times. The values p_i $(1 \leq i \leq n)$ are sampled from the uniform distribution $U[1, \alpha]$, where $\alpha = 10$ for Ins^S and $\alpha = 100$ for Ins^L. For each subset, we generate instances with $|N| = n = 10$, 20, 30, 40 and 50 jobs. Release dates r_i are drawn from $U[0, \tau P]$, where $P = \sum_{i \in N} p_i$ and $\tau \in \{0.0, 0.5, 1.0\}$. Due dates d_i are generated from $U[r_i+p_i, r_i+p_i+\rho P]$ with $\rho \in \{0.05, 0.25, 0.50\}$ and weights w_i stem from $U[1, 10]$. Up to here our generation is based on the instance generation procedure of Tanaka and Fujikuma (2012). Our modifications pertain to the generation of deadlines and precedence relations among jobs. Deadlines are chosen from $U[d_i, d_i + \phi P]$ with $\phi \in \{1.00, 1.25, 1.50\}$.

The addition of precedence constraints may lead to the generation of many instances with no feasible solution. For this reason, for each instance we first construct a feasible solution without considering precedence constraints (using the branch-and-bound algorithm proposed in Chapter 7). Next, the jobs are re-indexed according to the job order in this feasible solution. If no feasible solution exists even without precedence constraints, we use the original indices. Subsequently, a precedence graph is created using the RanGen software (Demeulemeester et al., 2003) with $OS \in \{0.00, 0.25, 0.50, 0.75\}$, where OS is the *order strength* of the graph (a measure for the density of the graph). For any instance, if a feasible solution exists without precedence constraints, then the addition of precedence constraints will never render it infeasible because RanGen only generates arcs from lower-indexed to higher-indexed jobs.

In conclusion, for each combination of $(\alpha, n, \tau, \rho, \phi, OS)$, four instances are generated; the total number of instances is thus $2 \times 5 \times 3 \times 3 \times 3 \times 4 \times 4 = 4320$.

α	Method	n		
		10	20	30
10	ASF	0.81	–	–
	ASF′	0.80	–	–
	TIF	0.43	2.02	53.47 (3)
	TIF′	0.64	2.97	88.17 (12)
100	ASF	0.92	–	–
	ASF′	0.95	–	–
	TIF	6.54	–	–
	TIF′	21.78	–	–

Table 5.2: Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 432) for the MIP formulations run on Ins with $n = 10$, 20 and 30.

5.5 Computational results

In our computational experiments, CPlex 12.3 is used to solve the MIP formulations. All computational results were obtained on a laptop Dell Latitude with 2.6 GHz Core(TM) i7-3720QM processor, 8GB of RAM, running under Windows 7. In all our experiments, the time limit is set to 1200 seconds. If an instance is not solved to guaranteed optimality, it is said to be 'unsolved' for the procedure. Throughout this section, we report averages computed only over the solved instances.

In Table 5.2 we compare the performance of the MIP formulations discussed in Section 5.3. In this table as well as in the following, we report the average runtime and the number of unsolved instances (if there are any). Based on Table 5.2, we conclude that the time-indexed formulations are far better than the assignment formulations when processing times are small. For large processing times, the performance of ASF is slightly better than TIF. Although ASF′ and TIF′ are tighter than their counterparts with aggregate precedence constraints, the extra computational effort needed to process the larger models increases CPU times in both TIF′ and ASF′.

We notice that none of the formulations is capable of solving instances larger than 30 jobs. In fact, if processing times are large, then none of the formulations is capable of solving instances larger than 10 jobs.

5.6 Summary and conclusion

In this chapter, we introduced a single-machine scheduling problem with total weighted tardiness penalties. We work with a rather general problem statement, in that both precedence constraints as well as time windows (release dates and deadlines) are part of the input. This generalizes quite a number of problems for which computational procedures have already been published. We propose an assignment MILP formulation and a time-indexed MILP formulation than can solve the problem (ASF and TIF). For each of these formulation, we provide a theoretically stronger alternative (ASF′ and TIF′). ASF and ASF′ are able to solve instances of maximum 10 jobs whereas TIF and TIF′ are able to solve instances of maximum 30 jobs (only when processing times are small). In general, these computational results suggest that none of the formulations is capable of solving medium-sized instances and thus studying approaches that solve the problem in much more efficient manner is promising.

Chapter 6

Lower bounds for GSMSP

To live a creative life we must first lose the fear of being wrong.

- Joseph Chilton Pearce

In the previous chapter, we introduced GSMSP which is a generic single-machine scheduling problem with total weighted tardiness penalties where the execution of each job is constrained with time windows and precedence constraints. In this chapter, we introduce valid lower bounds for GSMSP that can be implemented in exact algorithms. Section 6.1 introduces a conceptual formulation for our problem and Section 6.2 describes a very fast and trivial lower bounding procedure. In Section 6.3, we describe several lower bounds based on Lagrangian relaxation. In Section 6.4, we compare the quality of our proposed lower bounds and finally in Section 6.5, we provide a summary and some conclusions.

6.1 Another conceptual formulation

Let variable C_j denote the completion time of job $j \in N$ and let variable T_j represent the tardiness of job j. An alternative formulation of our problem is given by

$$\text{GSMSP}: \quad \min \quad \sum_{j=1}^{n} w_j T_j \tag{6.1}$$

subject to

$$T_j \geq C_j - d_j \qquad \forall j \in N \tag{6.2}$$
$$C_j \geq r_j + p_j \qquad \forall j \in N \tag{6.3}$$
$$C_j \leq \delta_j \qquad \forall j \in N \tag{6.4}$$
$$C_j \geq C_i + p_j \qquad \forall (i,j) \in A \tag{6.5}$$
$$C_j \geq C_i + p_j \text{ or } C_i \geq C_j + p_i \qquad \forall i,j \in N; i < j \tag{6.6}$$
$$T_j \geq 0 \qquad \forall j \in N \tag{6.7}$$
$$C_j \geq 0 \qquad \forall j \in N. \tag{6.8}$$

In the above formulation, constraints (6.2) and (6.7) reflect the definition of job tardiness. Constraints (6.3) and (6.4) enforce time windows. Constraints (6.5) ensure that each job is scheduled after all its predecessors. Constraints (6.6) guarantee that jobs do not overlap. We will use this formulation in Section 6.3 for producing lower bounds.

To the best of our knowledge, a lower-bound procedure specifically for GSMSP has to date not been developed in the literature. Lower bounds proposed for $1||\sum w_j T_j$, $1|prec|\sum w_j C_j$ and $1|r_j|\sum w_j C_j$, however, can also function as a lower bound for GSMSP; this is shown in the following theorems. These theorems are extensions of those presented in Akturk and Ozdemir (2000).

Let I be an instance of $1|\beta|\sum w_j T_j$. We construct an instance I' of $1||\sum w_j T_j$ by removing all constraints implied by β and an instance I'' of $1|\beta|\sum w_j C_j$ by replacing all due dates with zeros. Let $\text{TWT}^*(I)$ be the optimal objective value of I. Given any valid lower bound $lb_{I'}$ on the optimal value of I', we have:

Remark 6.1. $lb_{I'} \leq \text{TWT}^*(I)$.

Proof. Since $1||\sum w_j T_j$ is a relaxation of $1|\beta|\sum w_j T_j$, the optimal value of I' and any valid lower bound for this optimal value is considered as a valid lower bound for I. □

A job is called *early* if it finishes at or before its due date and is said to be *tardy* if it finishes after its due date. Let $C_j(S)$ be the completion time of job j in feasible solution S. For an optimal solution S^* to I, we partition N into two subsets: the set $\mathcal{E}$ of early jobs and the set $\mathcal{T}$ of tardy jobs. Let lb^E be a lower bound on the value $\sum_{j\in\mathcal{E}} w_j(d_j - C_j(S^*))$. Given any valid lower bound $\bar{lb}_{I''}$ on the optimal value of I'', we have:

Theorem 6.1. $\bar{lb}_{I''} - \sum_j w_j d_j + lb^E \leq \mathrm{TWT}^*(I)$.

Proof. The following equality holds:

$$\begin{aligned}\mathrm{TWT}^*(I) &= \sum_{j\in\mathcal{T}} w_j(C_j(S^*) - d_j) \\ &= \sum_{j\in N} w_j(C_j(S^*) - d_j) \\ &- \sum_{j\in\mathcal{E}} w_j(C_j(S^*) - d_j) = \sum_{j\in N} w_j C_j(S^*) \\ &- \sum_{j\in N} w_j d_j + \sum_{j\in\mathcal{E}} w_j(d_j - C_j(S^*)).\end{aligned}$$

Recall that $\bar{lb}_{I''}$ is a valid lower bound on the value $\sum_{j\in N} w_j C_j(S^*)$ and lb^E is a valid lower bound on the value $\sum_{j\in\mathcal{E}} w_j(d_j - C_j(S^*))$. □

In the following, we remove several combinations of constraints in GSMSP to construct subproblems for which there exist polynomial-time-bounded algorithms for computing lower bounds. These bounds then directly lead to valid lower bounds for GSMSP via Remark 6.1 and Theorem 6.1.

6.2 A trivial lower bound

Let $\mathrm{P_T}$ be the trivial subproblem of GSMSP in which constraints (6.2), (6.3), (6.4) and (6.5) are removed, which is then equivalent to $1||\sum w_j C_j$. An optimal solution S^* to $\mathrm{P_T}$ (with the optimal value $\mathrm{OPT}(S^*)$) follows sequence σ_T, which sequences jobs according to the *shortest weighted processing time* (SWPT) rule (Pinedo, 2008). By Remark 6.1 and 6.1, $\mathrm{LB_T} = \mathrm{OPT}(S^*) - \sum_j w_j d_j + lb^E$ is a valid lower bound for GSMSP. We compute lb^E as the summation of the earliness values when each job is scheduled at its latest possible starting time. Note that if $r_j = d_j = 0$ for

all jobs j and σ_T does not violate any deadline nor precedence constraint, then σ_T is optimal to GSMSP and $\mathrm{OPT} = \mathrm{LB_T}$. In B&B algorithms, this situation frequently occurs when some jobs have already been scheduled.

6.3 Lagrangian-relaxation-based bounds

In this section, we use Lagrangian relaxation for computing various lower bounds. Let P_0 be the subproblem of GSMSP in which constraints (6.3), (6.4) and (6.5) are removed (this problem is equivalent to $1||\sum w_j T_j$). This problem is studied by Potts (1985) and is considered as our base problem. Let λ be a vector of Lagrangian multipliers. Potts (1985) obtain the following Lagrangian problem associated with P_0:

$$\mathrm{LR_{P_0}}: \quad L_0(\lambda) = \min \sum_{j=1}^{n} (w_j - \lambda_j) T_j + \sum_{j=1}^{n} \lambda_j (C_j - d_j)$$
$$\text{subject to constraints (6.6)–(6.8).}$$

Parameter λ_j is the Lagrangian multiplier associated with job j ($0 \leq \lambda_j \leq w_j$). Potts and Van Wassenhove propose a polynomial-time algorithm to set the multipliers. Their algorithm yields a very good lower bound for P_0; they compute the optimal values of the multipliers in $\mathrm{O}(n \log n)$ time, and for a given set of multipliers, the bound itself can be computed in linear time. Let λ_{PV} be the best Lagrangian multipliers computed by Potts (1985); we refer to this lower bound as $\mathrm{LB_0} = L_0(\lambda_{\mathrm{PV}})$. By Remark 6.1, $\mathrm{LB_0}$ is also a valid bound for GSMSP. Quite a number of aspects of the definition of GSMSP are completely ignored in $\mathrm{LB_0}$, however; in the following sections, we will examine a number of ways to strengthen $\mathrm{LB_0}$.

6.3.1 Retrieving precedence constraints

When $A \neq \emptyset$ then incorporating some or all of precedence constraints into the lower bound will improve its quality. We partition arc set A as $A = A' \cup A''$, where $G' = (N, A')$ is a two-terminal *vertex serial-parallel* (VSP) graph and $G'' = (N, A'')$. Figure 6.1 depicts an example of this graph decomposition. For the precise definition of VSP graphs, we refer to Valdes et al. (1982). It should be noted that there exist two types of serial-parallel graphs: VSP graphs and *edge serial-parallel* (ESP) graphs. Valdes et al. (1982) describe the link between these two types: a graph is VSP if and only if its so-called 'line-digraph inverse' is ESP.

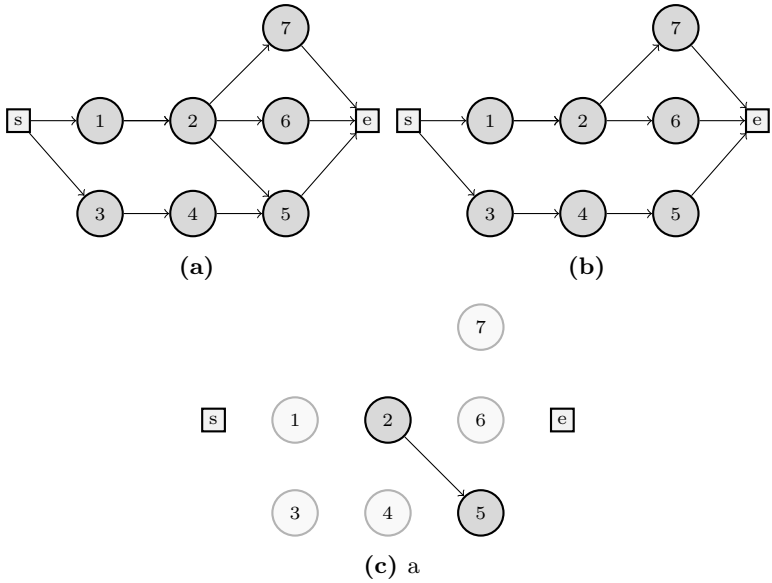


Figure 6.1: This figure shows (a) an example graph G, (b) an associated VSP sub-graph G' and (c) G''.

We split the set of constraints (6.5) into two subsets, as follows:

$$C_j \geq C_i + p_j \qquad \forall (i,j) \in A' \tag{6.9}$$

$$C_j \geq C_i + p_j \qquad \forall (i,j) \in A'' \tag{6.10}$$

We introduce P_1, which is a generalization of P_0 where precedence constraints are retrieved by imposing constraints (6.9) and (6.10). We create the following associated Lagrangian problem:

$$\mathrm{LR}_{\mathrm{P}_1}: \quad L_1(\lambda,\mu) = \min \sum_{j\in N} (w_j - \lambda_j) T_j + \sum_{j\in N} \lambda_j (C_j - d_j) + \sum_{j\in N} \sum_{k\in \mathcal{Q}_j} \mu_{jk}(C_j + p_k - C_k)$$

subject to constraints (6.6)–(6.9).

Here $\lambda_j \geq 0$ is again the multiplier associated with job j and $\mu_{jk} \geq 0$ denotes the Lagrangian multiplier associated with the arc $(j,k) \in A$. We deliberately keep constraints (6.9) in the Lagrangian problem $\mathrm{LR}_{\mathrm{P}_1}$. The objective function can be rewritten as

$$\sum_{j \in N} (w_j - \lambda_j) T_j + \sum_{j \in N} w'_j C_j + c$$

where

$$w'_j = \lambda_j + \sum_{k \in \mathcal{Q}_j} \mu_{jk} - \sum_{k \in \mathcal{P}_j} \mu_{kj}$$

and

$$c = \sum_{j \in N} \sum_{k \in \mathcal{Q}_j} \mu_{jk} p_k - \sum_{j \in N} \lambda_j d_j,$$

so it can be seen that $\mathrm{LR}_{\mathrm{P}_1}$ is a total-weighted-completion-times problem with serial-parallel precedence constraints, because all T_j will be set to zero and $\sum_{j \in N} (w_j - \lambda_j) T_j$ can be removed from the formulation. Lawler (1978) proposes an algorithm that solves this problem in $\mathrm{O}(n \log n)$ time provided that a *decomposition tree* is also given for the VSP graph G'. Valdes et al. (1982) propose an $\mathrm{O}(n+m)$-time algorithm to construct a decomposition tree of a VSP graph, where m is the number of arcs in the graph. Calinescu et al. (2012) show that any VSP graph (directed or undirected), including G', has at most $2n-3$ arcs. Therefore, for any given λ and μ, the problem $\mathrm{LR}_{\mathrm{P}_1}$ is solvable in $\mathrm{O}(n \log n)$ time. From the theory of Lagrangian relaxation (Fisher, 1981), for any choice of non-negative multipliers, $L_1(\lambda, \mu)$ provides a lower bound for P_1. By Remark 6.1, this lower bound is also valid for GSMSP. In Section 6.3.2, we explain how to choose appropriate values for λ and μ and Section 6.3.3 describes how to select a suitable VSP graph G' and how to construct a decomposition tree for G'.

6.3.2 Multiplier adjustment

We present a two-phase adjustment (TPA) procedure for the multipliers in $L_1(\lambda, \mu)$. Let λ_{TPA} and μ_{TPA} be Lagrangian multipliers adjusted by TPA; these lead to a new lower bound $\mathrm{LB}_1 = L_1(\lambda_{\mathrm{TPA}}, \mu_{\mathrm{TPA}})$. The TPA procedure is heuristic, in the sense that it may not minimize L_1 in λ and μ.

$k_{\max}$	n		
	20	30	40
0	11.576	8.505	6.85
5	14.579	17.454	13.493
10	15.026	18.147	14.065
20	15.207	18.419	14.344
50	15.296	18.503	14.466
100	15.310	18.508	14.495
∞	15.314	18.512	14.506

Table 6.1: The average percentage deviation between LB_1 and LB_0 tested on Ins^L.

In the first stage of TPA, we simply ignore precedence constraints altogether. For a feasible solution S, consider the function $g(\lambda, \mu, S)$ defined as follows:

$$g(\lambda, \mu, S) = \sum_{j \in N} (w_j - \lambda_j) T_j + \sum_{j \in N} \lambda_j (C_j - d_j) + \sum_{j \in N} \sum_{k \in \mathcal{Q}_j} \mu_{jk} (C_j + p_k - C_k).$$

We start with the Lagrangian problem $\hat{\mathrm{LR}}_{\mathrm{P}_1}$ where $\hat{L}_1(\lambda, \mu) = \min g(\lambda, \mu, S)$ subject to constraints (6.6)–(6.8), which is a relaxation of $\mathrm{LR}_{\mathrm{P}_1}$. We simply set all μ_{jk} to zero ($\mu = \mu_0 = (0, \ldots, 0)$); with this choice, $\hat{L}_1(\lambda, \mu) = L_0(\lambda)$ and we set $\lambda_{\mathrm{TPA}} = \lambda_{\mathrm{PV}}$.

In the second stage of TPA, the multipliers μ_{jk} are adjusted assuming that $\lambda = \lambda_{\mathrm{TPA}}$ is predefined and constant. This adjustment is an iterative heuristic; we adopt the *quick ascent direction* (QAD) algorithm proposed by van de Velde (1995). One iteration of TPA runs in $\mathrm{O}(m + n \log n)$ time, where $m = |A|$. We have run a number of experiments to evaluate the improvement of the lower bound as a function of the number of iterations $k_{\max}$. For a representative dataset, Table 6.1 shows that the average percentage deviation of LB_1 from LB_0 significantly increases in the first iterations, whereas after about five iterations the incremental improvement becomes rather limited; more information on the choices for $k_{\max}$ follows in Sections 6.3.3 and 6.4. The instance generation scheme is explained in Section 5.4.

Theorem 6.2. $\mathrm{LB}_0 \leq \mathrm{LB}_1$.

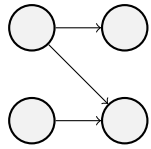


Figure 6.2: The forbidden subgraph for VSP graphs.

Proof. We argue that

$$\begin{aligned} \mathrm{LB}_0 = L_0(\lambda_{\mathrm{PV}}) = \hat{L}_1(\lambda_{\mathrm{PV}}, \mu_0) &\leq \hat{L}_1(\lambda_{\mathrm{TPA}}, \mu_{\mathrm{TPA}}) \\ &\leq L_1(\lambda_{\mathrm{TPA}}, \mu_{\mathrm{TPA}}) = \mathrm{LB}_1. \end{aligned}$$

The first inequality follows from Theorem 3 in van de Velde (1995), where it is shown that TPA generates a series of monotonically increasing lower bounds. The second inequality corresponds with Remark 6.1. □

6.3.3 Finding a VSP graph

LB_1 requires a decomposition of graph G into two subgraphs $G' = (N, A')$ and $G'' = (N, A'')$, such that $A' \cup A'' = A$ and G' is a VSP graph. The more arcs we can include in A', the tighter the lower bound. In the following, we discuss procedures to find a VSP subgraph G' with maximum number of arcs; we refer to this problem as the *maximum VSP subgraph* (MVSP) problem.

Valdes et al. (1982) state the following result:

Lemma 6.1. *A graph G is VSP if and only if its transitive closure does not contain the graph of Figure 6.2 as a subgraph.*

Valdes et al. refer to the pattern in Figure 6.2 as the *forbidden subgraph*. Polynomial-time exact procedures exist for finding an ESP subgraph with maximum number of nodes (Bein et al., 1992), but to the best of our knowledge, no exact approach for MVSP has been proposed yet in literature. McMahon and Lim (1993) suggest a heuristic traversal procedure to find and eliminate all forbidden subgraphs and, at the same time, construct a binary decomposition tree for the resulting VSP graph. Their procedure runs in $\mathrm{O}(n + m)$ time. The number of arcs in a VSP graph is bounded by $2n - 3$ for an arbitrary non-VSP graph, but the maximum number of arcs for an arbitrary input graph is $\mathrm{O}(n^2)$. We implement a

slightly modified variant of the algorithm in McMahon and Lim (1993) to compute G'; we select arcs for removal so that the lower bound remains reasonably tight. Simultaneously, it constructs a decomposition tree for the obtained VSP graph. The time complexity of $O(n+m)$ is maintained.

The structure of our heuristic decomposition and arc-elimination procedure is described in the following lines. The procedure constructs a decomposition tree by exploiting *parallel* and *serial node reduction* (Lawler, 1978). Parallel reduction merges a job pair into one single job if both jobs have the same predecessor and successor sets. In the decomposition tree, such jobs are linked by a P node, which means they can be processed in parallel (see Figure 6.3(b)). Serial reduction merges a job pair $\{i, j\}$ into one single job if arc $(i, j) \in A$, job i has only one successor and job j has only one predecessor. In the decomposition tree, such two jobs are linked by an S node, which means they cannot be processed in parallel (see Figure 6.3(d)). Whenever a forbidden subgraph is recognized, the procedure removes arcs such that the forbidden subgraph is resolved (removed) and the total number of removed arcs (including transitive and merged arcs) is approximately minimized (see Figures 6.3(b) and 6.3(c)). Notice that some arcs may actually represent multiple merged arcs, so removing one arc in one iteration might imply the removal of multiple arcs simultaneously in the original network G.

In the branch and bound tree of Chapter 7, the proposed algorithm is run only once, in the root node of the search tree. In each other node of the search tree, graphs G' and G'' are constructed by removing from the corresponding graphs in the parent node the arcs associated with the scheduled jobs; the resulting graphs are then the input for computing LB_1. Notice that for each child node, both graphs G' and G'' as well as the associated decomposition tree can be constructed in $O(n)$ time.

To evaluate the impact of our arc elimination procedure on the quality of the bounds, we examine two variations of LB_1, namely $\mathrm{LB}_1(\mathrm{VSP}) = L_1(\lambda_{\mathrm{TPA}}, \mu_{\mathrm{TPA}})$, where all forbidden graphs in G are resolved using the arc elimination procedure, and $\mathrm{LB}_1(\mathrm{NO}) = \hat{L}_1(\lambda_{\mathrm{TPA}}, \mu_{\mathrm{TPA}})$, in which we simply remove all arcs ($A' = \emptyset$ and $A'' = A$). Let $k_{\max}$ be the maximum number of iterations for TPA, as explained in Section 6.3.2. Table 6.2 demonstrates the success of our proposed algorithm in tightening the bound. The distance between the bounds is decreasing with increasing $k_{\max}$, but in a B&B algorithm, a large value for $k_{\max}$ becomes computationally prohibitive.

Theorem 6.3. $\mathrm{LB}_1(\mathrm{NO}) \leq \mathrm{LB}_1(\mathrm{VSP})$ *for the same value of* $k_{\max}$.

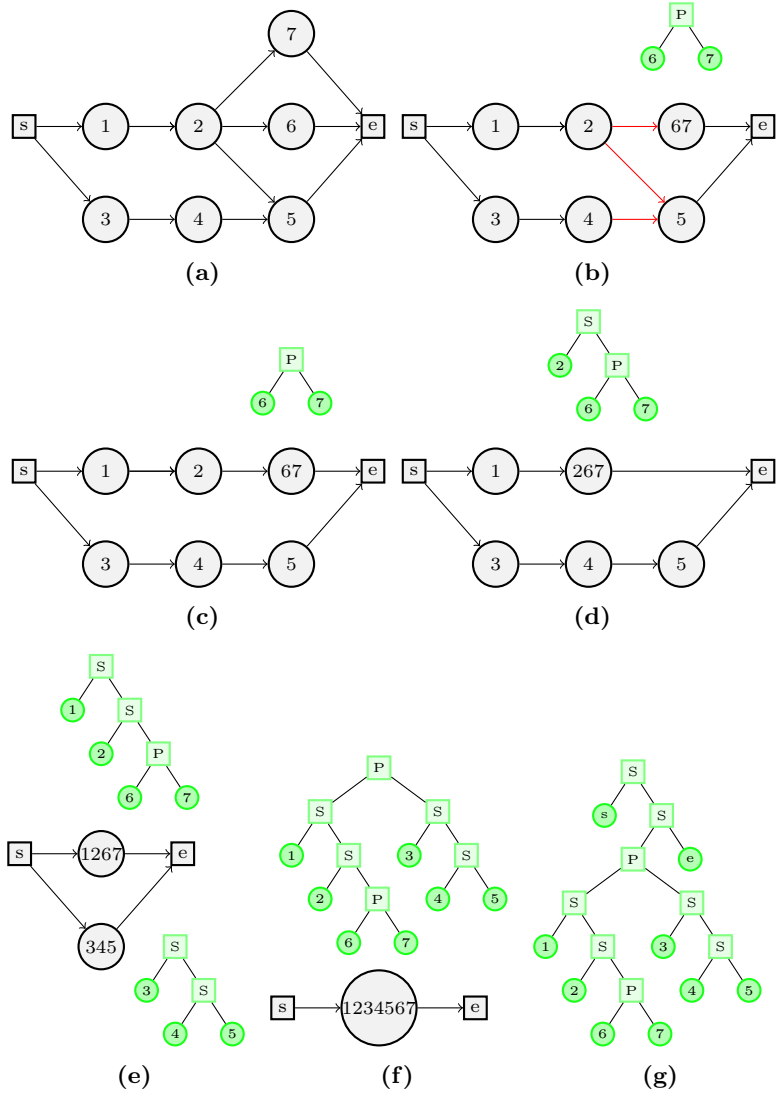


Figure 6.3: Modified traversal algorithm applied to the input graph in (a).

$k_{\max}$	LB_1(VSP)	LB_1(NO)
0	6.850	0
1	10.515	9.057
2	12.171	11.497
3	12.896	12.538
5	13.493	13.385
10	14.344	14.020
100	14.466	14.458

Table 6.2: The average percentage deviation between LB_1 and LB_0 tested on Ins^L with 40 jobs.

Proof. LB_1(NO) is obtained by solving $\mathrm{LR}_{\mathrm{P}_1}$ with $A' = \emptyset$ and $A'' = A$, so with the same multipliers the problem associated with LB_1(NO) is a relaxation of the problem associated with LB_1(VSP). The multipliers are updated with TPA in both cases, and will indeed be the same for a given $k_{\max}$, so the theorem holds. □

Bound LB_1 turns out not to be very tight when release dates are rather heterogeneous. Below, we examine two means to produce a stronger bound, namely block decomposition and job splitting.

6.3.3.1 Block decomposition

We follow Hariri and Potts (1983), Pan and Shi (2005) and Potts and Van Wassenhove (1983) in setting up a decomposition of the job set into separate *blocks*: a *block* is a subset of jobs for which it is a dominant decision to schedule them together. We sort and renumber all jobs in non-decreasing order of their modified release dates $\bar{r}_j$ (that are computed by applying precedence constraints propagation); as a tie-breaking criterion, we consider non-increasing order of w_j/p_j. The resulting *non-delay* sequence of jobs is given by $\sigma^r = (1, \ldots, n)$, where a sequence is said to be 'non-delay' if the machine is never kept idle while some jobs are waiting to be processed (Pinedo, 2008). Let $B_i = (u_i, \ldots, v_i)$ be one block (in which jobs are sorted according to their new indices). The set $B = \{B_1, \ldots, B_\kappa\}$ is a *valid* decomposition of the job set into κ blocks if the following conditions are satisfied:

1. $u_1 = 1$;

2. for each i, j with $1 < i \leq \kappa$ and $1 \leq j \leq n$, if $u_i = j$ then $v_{i-1} = j-1$ and vice versa;
3. for each i, j with $1 \leq i \leq \kappa$ and $u_i \leq j \leq v_i$, we have $\bar{r}_{u_i} + \sum_{s=u_i}^{j-1} p_s \geq \bar{r}_j$.

Although the sequencing of the jobs within one block is actually still open, the sequencing of the blocks is pre-determined. Given a valid set of blocks B, we compute LB_1 for each block $B_i \in B$ separately. The value LB_2 is then the sum of the bounds per block; analogously to Hariri and Potts (1983), Pan and Shi (2005) and Potts and Van Wassenhove (1983), LB_2 can be shown to be a lower bound for GSMSP.

We define $\mathrm{LB}_1^* = L_1(\lambda^*, \mu^*)$, where λ^* and μ^* are optimal choices for the Lagrangian multipliers for LB_1, and LB_2^*, which is computed by adding the contribution $L_1(\lambda^*_{B_i}, \mu^*_{B_i})$ for each block B_i, where $\lambda^*_{B_i}$ and $\mu^*_{B_i}$ are the optimal choices for the multipliers for block B_i.

Theorem 6.4. $\mathrm{LB}_1^* \leq \mathrm{LB}_2^*$.

Proof. We introduce $g_{B_i}(\lambda, \mu, S)$ as follows:

$$g_{B_i}(\lambda, \mu, S) = \sum_{j \in B_i} (w_j - \lambda_j) T_j + \sum_{j \in B_i} \lambda_j (C_j - d_j) + \sum_{j \in B_i} \sum_{k \in \mathcal{Q}_j} \mu_{jk} (C_j + p_k - C_k).$$

Let S_1^* be an optimal solution to LB_1^* and $S_2^* = (S^*_{B_1}, \ldots, S^*_{B_\kappa})$ an optimal solution to LB_2^*. The following result is derived:

$$\begin{aligned} \mathrm{LB}_1^* &= g(\lambda^*, \mu^*, S_1^*) \leq g(\lambda^*, \mu^*, S_2^*) \\ &= \sum_{i=1}^{\kappa} g_{B_i}(\lambda^*, \mu^*, S^*_{B_i}) \leq \sum_{i=1}^{\kappa} g_{B_i}(\lambda^*_{B_i}, \mu^*_{B_i}, S^*_{B_i}) = \mathrm{LB}_2^*. \end{aligned}$$

□

Although TPA might not find $\lambda^*_{B_i}$ and $\mu^*_{B_i}$ and thus the same result as Theorem 6.4 might not hold for LB_1 and LB_2, empirical results show that LB_2 is on average far tighter than LB_1 (these results are shown in Table 6.3).

6.3.3.2 Job splitting

It sometimes happens that the decomposition procedure fails to improve the bound (only one block is created and $\mathrm{LB}_2 = \mathrm{LB}_1$). Another approach is to explicitly re-introduce the release-date constraints (which have been removed previously). We define problem P_2, which is a generalization of P_1 in which the release-date constraints (6.3) are included. The associated Lagrangian problem is

$$\mathrm{LR}_{\mathrm{P}_2}: \quad L_2(\lambda,\mu) = \min \sum_{j\in N} w'_j C_j + c$$
$$\text{subject to constraints (6.3),(6.6)–(6.8).}$$

Contrary to $\mathrm{LR}_{\mathrm{P}_1}$, we now remove the serial-parallel precedence constraints because they render the Lagrangian problem too difficult. Problem $\mathrm{LR}_{\mathrm{P}_2}$ is a total-weighted-completion-times problem with release dates. This problem is known to be NP-hard (Lenstra et al., 1977), but a number of efficient polynomial algorithms, which are based on job splitting, have been proposed to compute tight lower bounds (Belouadah et al., 1992; Hariri and Potts, 1983; Nessah and Kacem, 2012). One of these algorithms is the SS procedure proposed by Belouadah et al. (1992), which runs in $\mathrm{O}(n \log n)$ time and which we adopt here. Essentially, we again decompose the job set into a set of blocks B and compute $L_2(\lambda,\mu)$ for each block $B_i \in B$. The lower bound $\mathrm{LB}_2^{\mathrm{SS_r}}$ is again the sum of the contributions of the individual blocks. Experiments show that $\mathrm{LB}_2^{\mathrm{SS_r}}$ is typically tighter than LB_2 when the release dates are unequal. With equal release dates, on the other hand, normally $\mathrm{LB}_2 \geq \mathrm{LB}_2^{\mathrm{SS_r}}$ because LB_2 incorporates a part of the precedence graph. TPA is applied also here for multiplier updates.

We introduce P'_2, which is a generalization of P_1 where deadline constraints are retrieved by inclusion of the constraint set (6.4). The associated Lagrangian problem is

$$\mathrm{LR}_{\mathrm{P}'_2}: \quad L'_2(\lambda,\mu) = \min \sum_{j\in N} w'_j C_j + c$$
$$\text{subject to constraints (6.4),(6.6)–(6.8).}$$

$\mathrm{LR}_{\mathrm{P}'_2}$ is a total-weighted-completion-times problem with deadlines. This problem is known to be NP-hard (Lenstra et al., 1977). Posner (1985) proposes a job-splitting lower bounding scheme for $\mathrm{LR}_{\mathrm{P}'_2}$ that uses $\mathrm{O}(n \log n)$

time; the lower bound $\mathrm{LB}_2^{\mathrm{SS}_\delta}$ results from block decomposition and computation of $L_2'(\lambda, \mu)$ for each block. We again apply TPA for setting the multiplies.

6.3.4 Improvement by slack variables

Relaxed inequality constraints can be considered to be 'nasty' constraints because they decrease the quality of lower bounds. We follow Hoogeveen and van de Velde (1995) in exploiting the advantages of slack variables to lessen the effect of such nasty constraints to improve the quality of the lower bounds.

We introduce two non-negative vectors of slack variables: vector $\mathbf{y} = (y_1, \ldots, y_n)$ and vector $\mathbf{z} = (z_{11}, \ldots, z_{1n}, \ldots, z_{n1}, \ldots, z_{nn})$. Consider the following sets of constraints:

$$T_j = C_j - d_j + y_j \qquad \forall j \in N \tag{6.11}$$

$$C_j = C_i + p_j + z_{ij} \qquad \forall (i,j) \in A \tag{6.12}$$

$$y_j, z_{ij} \geq 0 \qquad \forall i, j \in N \tag{6.13}$$

Let problem P_3 be the variant of problem P_1 in which the sets of constraints (6.2) and (6.5) are replaced by the constraints (6.11)–(6.13). The Lagrangian problem associated with P_3 is

$$\mathrm{LR}_{\mathrm{P}_3}: \quad L_3(\lambda, \mu) = \min \sum_{j=1}^{n} (w_j - \lambda_j) T_j + \sum_{j=1}^{n} \lambda_j y_j + \sum_{j=1}^{n} \sum_{k \in \mathcal{Q}_j} \mu_{jk} z_{jk} + \sum_{j \in N} w_j' C_j + c$$

subject to constraints (6.6)–(6.9) and (6.13).

The values of the variables T_j, y_j and z_{jk} are zero in any optimal solution to $\mathrm{LR}_{\mathrm{P}_3}$ because for $i, j \in N$ the following inequalities hold: $0 \leq \lambda_j \leq w_j$ and $\mu_{jk} \geq 0$. In an optimal solution to P_3, however, these values might not be zero. In fact, according to the set of constraints (6.11), unless $C_j = d_j$, either T_j or y_j is nonzero. Also, from constraints (6.12), z_{jk} may not be zero when job j has at least two successors or job k has at least two predecessors in G. We introduce three problems that each carry a part of the objective function of $\mathrm{LR}_{\mathrm{P}_3}$, one of which is $\mathrm{LR}_{\mathrm{P}_1}$ and the other two are the following two slack-variable (SV) problems, where Y is

the set of all **y**-vectors corresponding to feasible solutions to P_3 and Z similarly contains all **z**-vectors.

$$P_{SV1}: \quad SV_1(\lambda) = \min \sum_{j=1}^{n} (w_j - \lambda_j)T_j + \sum_{j=1}^{n} \lambda_j y_j$$

subject to constraints (6.6),(6.7),(6.9) and $\mathbf{y} \in Y$;

$$P_{SV2}: \quad SV_2(\mu) = \min \sum_{j=1}^{n} \sum_{k \in \mathcal{Q}_j} \mu_{jk} z_{jk}$$

subject to constraint $\mathbf{z} \in Z$.

Note that the term $\sum_{j=1}^{n} (w_j - \lambda_j)T_j$ appears in two of the problems, but it will be set to zero anyway in LR_{P_1}.

Hoogeveen and van de Velde (1995) propose $O(n \log n)$-time procedures to compute valid lower bounds for P_{SV1} and P_{SV2}. Let $LB_{SV1} \geq 0$ and $LB_{SV2} \geq 0$ be lower bounds for P_{SV1} and P_{SV2}, respectively. By adding LB_{SV1} and LB_{SV2} to LB_2, a better lower bound LB_3 for P is obtained (Hoogeveen and van de Velde, 1995). The same SV problems can also be constructed for $LB_2^{SS_r}$ and $LB_2^{SS_\delta}$ to lead to bounds $LB_3^{SS_r} = LB_2^{SS_r} + LB_{SV1} + LB_{SV2}$ and $LB_3^{SS_\delta} = LB_2^{SS_\delta} + LB_{SV1} + LB_{SV2}$. We have the following result:

Observation 6.1. $LB_2 \leq LB_3$, $LB_2^{SS_r} \leq LB_3^{SS_r}$ *and* $LB_2^{SS_\delta} \leq LB_3^{SS_\delta}$.

6.3.5 Other Lagrangian bounds

All the lower bounds introduced in this section are based on the formulation (6.1)-(6.8). Other Lagrangian-relaxation-based lower bounds have also been proposed for special cases of this problem. These other bounds are mostly based on other (conceptual) formulations. For example, to achieve a lower bound, Lagrangian penalties could be added to the objective function while allowing jobs to be processed repeatedly. Many variants of such a lower bound exist (Tanaka and Fujikuma, 2012; Tanaka and Sato, 2013; Tanaka et al., 2009), but most of these variants are either too weak or too slow. Another lower bound based on Lagrangian relaxation is obtained by relaxing the capacity constraints, such that jobs are allowed to be processed in parallel in exchange for Lagrangian penalties (Tang et al., 2007).

6.4 The quality of the lower bounds

We compare the quality of the lower bounds for the subset of instances with large processing times and $n = 30$. We set $k_{\max} = 10$ for all lower bounds. The detailed results of this comparison are reported in Table 6.3.

The average gap for LB_1 is less than or equal to that for LB_0, especially when the precedence graph is dense; for $OS = 0$, on the other hand, there are no precedence constraints and LB_0 and LB_1 are essentially the same. A similar observation can be made for LB_1 and LB_2, where the gap for LB_2 is noticeably smaller than that for LB_1 when release dates are imposed, while in the case $\tau = 0$, only one block is created and the lower bounds LB_1 and LB_2 coincide. The average gap for LB_3 is indeed smaller than that for LB_2, as was to be expected according to Observation 6.1. Despite all the improvements, it should be noted that all gaps remain relatively large.

Although we have no theoretical result that would indicate a better performance of $\mathrm{LB}_2^{\mathrm{SS_r}}$ in comparison with LB_2, the average gap for $\mathrm{LB}_2^{\mathrm{SS_r}}$ is less than that for LB_2 in case of non-zero release dates. When release dates are zero, however, the gap for $\mathrm{LB}_2^{\mathrm{SS_r}}$ is larger than or equal to that for LB_2. In fact, when release dates are zero, only one block is created and constraints (6.3) can be removed from $\mathrm{LR}_{\mathrm{P}_2}$, and thus $\mathrm{LR}_{\mathrm{P}_2}$ is a relaxation of $\mathrm{LR}_{\mathrm{P}_1}$. $\mathrm{LB}_2^{\mathrm{SS}_\delta}$ performs better than LB_2 and $\mathrm{LB}_2^{\mathrm{SS_r}}$ for most of the instances. The gap for $\mathrm{LB}_3^{\mathrm{SS_r}}$ is less than that for $\mathrm{LB}_2^{\mathrm{SS_r}}$ and a similar observation holds for $\mathrm{LB}_3^{\mathrm{SS}_\delta}$ versus $\mathrm{LB}_2^{\mathrm{SS}_\delta}$, which confirms the result in Observation 6.1.

		LB_0	LB_1	LB_2	$LB_2^{SS_r}$	$LB_2^{SS_\delta}$	LB_3	$LB_3^{SS_r}$	$LB_3^{SS_\delta}$	LB_{Best}
OS	0.00	50.505	50.505	44.369	43.313	36.857	43.142	42.086	35.630	35.372
	0.25	67.776	63.465	53.444	52.702	52.300	51.955	51.013	50.568	49.834
	0.50	71.461	66.108	52.378	51.890	52.021	51.216	50.727	50.767	50.352
	0.75	77.055	69.836	50.769	50.520	50.565	49.430	49.182	49.041	48.863
τ	0.00	38.141	29.169	29.169	29.182	24.784	28.152	28.165	23.667	23.667
	0.50	76.712	73.157	59.554	58.724	57.699	57.876	57.046	55.922	55.656
	1.00	85.704	85.539	62.494	61.255	61.655	61.117	59.878	60.233	59.309
All	-	67.161	62.911	50.638	49.950	48.268	49.275	48.586	46.824	46.425

Table 6.3: Average percentage gap from optimal value.

6.5 Summary and conclusion

In this chapter we propose a number of lower bounding approaches that are based Lagrangian relaxation. These lower bounds are obtained by removing different combination of constraints and then gradually incorporating them via Lagrangian penalties. Among our proposed lower bounding approaches, $LB_3^{SS_\delta}$ performs the best in terms of average percentage gap from optimal value. In spite of the fact that our proposed lower bounds are meant to deal with instances with non-zero release date and with precedence relations, their performances seem to be weak (the average percentage gap from optimal value is on average more than 50%) when release dates are loose and/or precedence graph is dense. This suggests that in future researches interested authors must focus on finding lower bounding approaches that produce tight bounds specially when release dates are loose and/or precedence graph is dense.

Chapter 7

A branch and bound algorithm for GSMSP

Arriving at one goal is the starting point to another.

- John Dewey

In the previous chapters, we introduced GSMSP and proposed several lower bounds. In this chapter, we present a branch-and-bound (B&B) algorithm that can solve small- and medium-sized instances of GSMSP until optimality. The remainder of this chapter is structured as follows. We first present two branching schemes in Section 7.1. Then we introduce six different dominance rules in Section 7.2. Next in Section 7.3, we propose an algorithm that computes an initial upper bound for GSMSP. After that in Section 7.4 we report detailed computational results and finally in Section 7.5 a summary is provided.

7.1 Branching strategies

In this section we discuss two different branching strategies for our B&B algorithm. The structure of the B&B search trees is as follows: each tree consists of a finite number of nodes and branches, and at each level of the tree we make a sequencing decision for one job. Each node thus

Figure 7.1: The structure of a partial schedule.

corresponds with a selection $S_P \subseteq N$ containing the already scheduled jobs and a set of unscheduled jobs $U = N \backslash S_P$. Each node also has two feasible partial sequences σ_B and σ_E of the scheduled jobs (each $i \in S_P$ appears in exactly one of these two): σ_B (respectively σ_E) denotes the partial sequence of jobs scheduled from the beginning (respectively end) of the scheduling horizon; see Figure 7.1 for an illustration. All jobs that are not scheduled, belong to the set of unscheduled jobs $U = E_B \cup E_E \cup E_N$. E_B is subset of unscheduled jobs that are eligible to be scheduled immediately after the last job in σ_B, E_E is the subset of unscheduled jobs that are eligible to be scheduled immediately before the first job in σ_E and E_N is the subset of unscheduled jobs that are not in $E_B \cup E_E$.

The root node represents an empty schedule ($S_P = \sigma_B = \sigma_E = \emptyset$). Each node branches into a number of child nodes, which each correspond with the scheduling of one particular job, called the *decision job*, as early as possible after the last job in σ_B or as late as possible before the first job in σ_E. A branch is called a *forward branch* if it schedules a job after the last job in σ_B, and is called a *backward branch* if it schedules a job before the first job in σ_E. In our branching strategies, there will be either only forward branches or only backward branches emanating from each given node. We will say that a node is of type FB (respectively BB) if all its branches are forward (respectively backward) branches.

Although scheduling jobs backward (from the end of the time horizon) often improves the tightness of lower bounds (Potts, 1985) when release dates are equal, it probably decreases the quality of the lower bounds in the presence of non-equal release dates; see Sections 6.3 and 7.1.2 for a description of backward branching and of the lower bounds, respectively, and Section 7.4.2 for the empirical results and a discussion. Also, the efficiency of some dominance rules may decrease when we switch from forward scheduling to backward scheduling; see Section 7.2.4 for more details. We propose two B&B algorithms, each applying one of the branching strategies: BB1 corresponds with branching strategy 1 where only FB nodes are used and BB2 corresponds with branching strategy 2 where both FB and

BB are created. The bounding and the dominance properties discussed in the following sections are the same in both B&B algorithms.

Let $C_{\max}(\sigma)$ be the completion time of the last job in the sequence σ. Throughout the branching procedure, we maintain two vectors of *updated release dates*, namely $\hat{\mathbf{r}} = (\hat{r}_1, \ldots, \hat{r}_n)$ and $\bar{\mathbf{r}} = (\bar{r}_1, \ldots, \bar{r}_n)$, defined as follows:

$$\hat{r}_j = \max\{r_j, C_{\max}(\sigma_B)\}$$
$$\bar{r}_j = \max\left\{\hat{r}_j, \max_{k \in \mathcal{P}_j}\{\bar{r}_k + p_k\}\right\}.$$

Let $st(\pi)$ denote the start time of the first job according to sequence π. In line with the two vectors of updated release dates, we also introduce two vectors of *updated deadlines*, namely $\hat{\delta} = (\hat{\delta}_1, \ldots, \hat{\delta}_n)$ and $\bar{\delta} = (\bar{\delta}_1, \ldots, \bar{\delta}_n)$, which are recursively computed as follows:

$$\hat{\delta}_j = \min\{\delta_j, st(\sigma_E)\}$$
$$\bar{\delta}_j = \min\left\{\hat{\delta}_j, \min_{k \in \mathcal{Q}_j}\{\bar{\delta}_k - p_k\}\right\}.$$

We use these updated release dates and deadlines in computing lower bounds and dominance rules. $\bar{\delta}$ and $\bar{\mathbf{r}}$ are more restrictive than $\hat{\delta}$ and $\hat{\mathbf{r}}$ in each node of the search tree ($\bar{r}_j \geq \hat{r}_j$ and $\bar{\delta}_j \leq \hat{\delta}_j$). Although being restrictive often is positive, $\hat{r}_j$ and $\hat{\delta}_j$ are occasionally preferred over $\bar{r}_j$ and $\bar{\delta}_j$, specifically in parts of computations related to the dominance rules discussed in Section 7.2. Further explanations of these occasions are given in Section 7.2. There are many cases in which $\bar{r}_j = \hat{r}_j$ (respectively $\bar{\delta}_j = \hat{\delta}_j$) and either of the updated release dates (respectively deadlines) can be used. In these cases, we use $\hat{r}_j$ (respectively $\hat{\delta}_j$) because less computations are needed.

7.1.1 Branching strategy 1

Branching strategy 1 only uses FB nodes. The search tree is explored depth-first such that among children of a node, those with larger out-degrees (number of transitive successors) of their decision jobs in $\hat{G}$ are visited first. As a tie-breaking rule, among children with equal out-degrees of their decision jobs, the node with lower index is visited first.

Example. *Figure 7.2 illustrates branching strategy 1 applied to the example in Section 5.2 ; an asterisk '*' indicates that the position has not been*

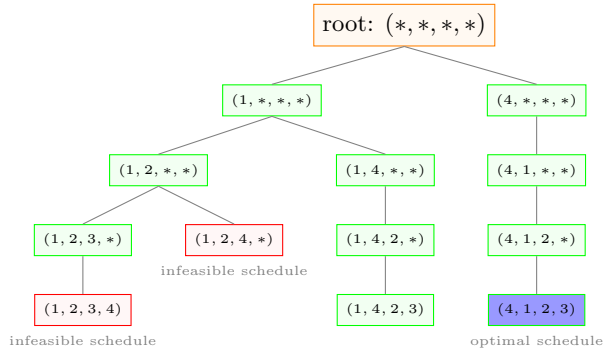


Figure 7.2: Branching strategy 1 for the example in Section 5.2 without dominance rules and without lower bounds.

decided yet. Among the children of the root node, the node $(1, *, *, *)$ *corresponds with the decision job (job* 1*) with the largest out-degree (namely* 3*). As a result, the node* $(1, *, *, *)$ *is visited first. The nodes* $(2, *, *, *)$ *and* $(3, *, *, *)$ *are not in the tree because they violate precedence constraints. Among the children of* $(1, *, *, *)$*, the node* $(1, 2, *, *)$ *is visited first because it has the decision job* 2 *with the largest out-degree. Among the children of* $(1, 2, *, *)$*, the node* $(1, 2, 3, *)$ *is visited first because its decision job has the largest out-degree and the smallest index. In Figure 7.2, green nodes are* FB *nodes; no* BB *nodes are present. Red nodes are considered infeasible because the completion of a job (namely job* 4*) occurs after its deadline. The node* $(1, 4, 2, 3)$ *corresponds with a feasible schedule, but it is not optimal: its objective value is greater than* 15*, which is attained by the optimal sequence* $(4, 1, 2, 3)$*.*

7.1.2 Branching strategy 2

In branching strategy 2, we try to exploit the advantages of backward scheduling whenever possible, so the search tree consists of both FB and BB nodes. If the inequality $C_{\max}(\sigma_B) < r_{\max}(U) = \max_{j \in U} \{r_j\}$ holds, then the start times of the jobs in σ_E will depend on the order in which unscheduled jobs are processed. Therefore, if the inequality $C_{\max}(\sigma_B) < r_{\max}(U)$ holds, the corresponding node is of type FB. Otherwise, the completion time of the last job in σ_E can be computed regardless of the

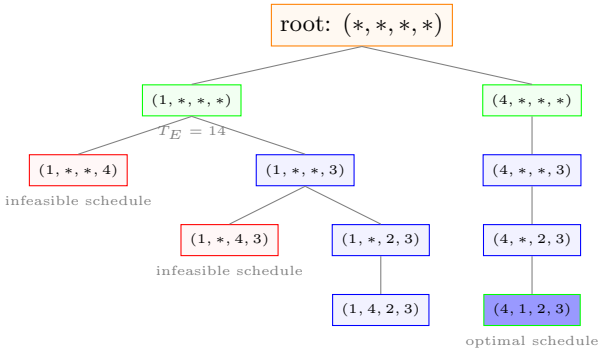


Figure 7.3: Branching strategy 2 for the example in Section 5.2 without dominance rules and without lower bounds.

sequencing decisions for the jobs in U, and we have a BB node. The branching is *depth-first* for both FB and BB nodes. Among the children of an FB (respectively BB) node, those with higher (respectively lower) out-degrees of their decision jobs are visited first. As a tie-breaking rule, among children with equal out-degrees, the node with lower (respectively higher) index is visited first.

Example. *Figure 7.3 illustrates branching strategy 2 for the example in Section 5.2 ; green nodes are of type* FB *and blue nodes are of type* BB*. The root node is* FB *because* $C_{\max}(\emptyset) = 0 < 4 = r_{\max}(\{1,2,3,4\})$*. At the node labeled* $(1,*,*,*)$*, the completion time* $C_{\max}(1,*,*,*) = 5$ *of the decision job surpasses* $r_{\max}(\{1,2,3,4\}) = 4$*, therefore the end of scheduling horizon is computed (*$T_E = 5+3+4+2 = 14$*) and the node is* BB*. The red nodes are infeasible because the completion time of job* 4 *falls after its deadline.*

7.2 Dominance properties

Our search procedure also incorporates a number of dominance rules, which will be described in this section. We will use the following additional notation. Given two partial sequences $\pi = (\pi_1, \ldots, \pi_k)$ and $\pi' = (\pi'_1, \ldots, \pi'_{k'})$, we define a merge operator as follows: $\pi|\pi' = (\pi_1, \ldots, \pi_k,$

$\pi'_1, \ldots, \pi'_{k'}$). If π' contains only one job j then we can also write $\pi|j = (\pi_1, \ldots, \pi_k, j)$, and similarly if $\pi = (j)$ then $j|\pi' = (j, \pi'_1, \ldots, \pi'_{k'})$.

7.2.1 General dominance rules

We use the lower bounds proposed in Chapter 6 to prune the search tree. Let LB(U) represent any of the lower bounds described in Chapter 6, applied to the set U of unscheduled jobs, and let S_{best} be the currently best known feasible solution. Notice that TWT(S_{best}) is an upper bound for TWT(S^*). The following dominance rule is then immediate:

Dominance rule 7.1 ($\mathrm{DR}_{7.1}$)**.** *Consider a node associated with selection S_P. If*

$$\mathrm{TWT}(S_P) + \mathrm{LB}(U) \geq \mathrm{TWT}(S_{best}),$$

then the node associated with S_P can be fathomed.

As we already introduced in Section 7.1, a partial schedule can be denoted by either S_P or (σ_B, σ_E). Multiple lower bounds can be used to fathom a node. The selection of lower bounds and the order in which they are computed obviously influences the performance of the B&B algorithm. These issues are examined in Section 7.4.

The subset of *active schedules* is dominant for total weighted tardiness problems (Conway et al., 1967; Pinedo, 2008). A feasible schedule is called *active* if it is not possible to construct another schedule by changing the sequence of jobs such that at least one job is finishing earlier and no other job finishes later. The dominance of active schedules holds even when deadlines and precedence constraints are given.

Dominance rule 7.2 ($\mathrm{DR}_{7.2}$)**.** *Consider a node associated with $(\sigma_B, \emptyset)$ that is selected for forward branching, and let j be a job belonging to E_B. If $\bar{r}_j \geq \min_{k \in E_B}\{\bar{r}_k + p_k\}$, then the child node associated with the schedule $(\sigma_B|j, \emptyset)$ can be fathomed.*

We also prune a branch whenever an obvious violation of the deadline constraints is detected. A partial schedule associated with a particular node is not always extended to a feasible schedule. Scheduling a job in one particular position may force other jobs to violate their deadline constraints, even though it does not violate its own constraints. Let $\mathcal{A}$ be an arbitrary subset of U and let $\Pi_{\mathcal{A}}$ be the set of all possible permutations of jobs in $\mathcal{A}$. The following theorem states when a job is scheduled in a 'wrong position', meaning that it will lead to a violation of deadline constraints.

Theorem 7.1. *Consider a partial schedule* (σ_B, σ_E). *If there exists any non-empty subset* $\mathcal{A} \subset U$ *such that the inequality* $\min_{\pi \in \Pi_{\mathcal{A}}}\{C_{\max}(\sigma_B|\pi)\} > \max_{j \in \mathcal{A}}\{\bar{\delta}_j\}$ *holds, then the schedule* (σ_B, σ_E) *is not feasible.*

Proof. If $\min_{\pi \in \Pi_{\mathcal{A}}}\{C_{\max}(\sigma_B|\pi)\}$ is greater than $\max_{j \in \mathcal{A}}\{\bar{\delta}_j\}$ then at least one job in $\mathcal{A}$ cannot be scheduled before its deadline and the schedule (σ_B, σ_E) is not feasible. □

The problem $\min_{\pi \in \Pi_{\mathcal{A}}}\{C_{\max}(\sigma_B|\pi)\}$, which equals $1|r_j, \delta_j, prec|C_{\max}$, is NP-hard because the mere verification of the existence of a feasible schedule is already NP-complete. We remove deadlines and create a new problem whose optimal solution is computed in $O(n^2)$ time (Lawler, 1973). For computational efficiency, we use a linear-time lower bound for this new problem. This lower bound is computed as follows: $\min_{j \in \mathcal{A} \cap E_B}\{\bar{r}_j\} + \sum_{j \in \mathcal{A}} p_j$.

Dominance rule 7.3 ($DR_{7.3}$)**.** *The node associated with* (σ_B, σ_E) *can be eliminated if at least one of the following conditions is satisfied:*

1. *if* $\sigma_E = \emptyset$ *and the condition of Theorem 7.1 is satisfied for the partial schedule* $(\sigma_B, \emptyset)$*;*
2. *if* $\sigma_E \neq \emptyset$ *and* $\max_{j \in U}\{\bar{\delta}_j\} < st(\sigma_E)$.

While additional precedence constraints could be added to the problem considering time windows and using constraint propagation techniques, the solution representation for our B&B algorithms and the above dominance rules ($DR_{7.2}$ and $DR_{7.3}$) are devised in such a way that any violation of these additional precedence constraints is dominated.

Example. *Consider two jobs* i *and* j *with* $p_i = p_j = 10$, $r_i = 0$, $r_j = 5$, $\delta_i = 20$ *and* $\delta_j = 30$. *Using constraint propagation techniques, it could be possible to include an extra constraint that allows the processing of job* j *to occur only after the completion of job* i. *Such an additional constraint is not necessary, however, because in the above-described situation, all sequences in which job* j *precedes job* i *will be automatically fathomed by* $DR_{7.2}$ *and* $DR_{7.3}$. *Moreover, increasing the density of the precedence graph in this way would also decrease the tightness of the lower bounds, which is undesirable.*

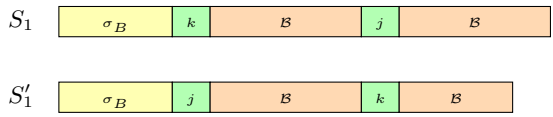


Figure 7.4: Schedules S_1 and S'_1.

7.2.2 Dominance rule based on two-job interchange

We describe a dominance rule based on job interchange. This dominance rule consists of two parts. The first part deals with the interchange of jobs in an FB node whereas the second part deals with the interchange of jobs in a BB node.

7.2.2.1 Interchanging jobs in an FB node

In an FB node, consider jobs $j, k \in E_B$ that are not identical (they differ in at least one of their parameters). We will always assume that $\hat{r}_k < \hat{r}_j + p_j$ and $\hat{r}_j < \hat{r}_k + p_k$, because otherwise Dominance rule 7.2 enforces the scheduling of the job with smaller $\hat{r}$ before the job with larger $\hat{r}$; note here that $\hat{r}_j = \bar{r}_j$ and $\hat{r}_k = \bar{r}_k$ because all predecessors of jobs j and k has already been scheduled and therefore the branching decisions cover the propagation of precedence constraints. We also assume that any successor of job k is also a successor of job j ($\mathcal{Q}_k \subset \mathcal{Q}_j$). Consider a node of the search tree in which job k is scheduled at or after the completion of sequence σ_B. Suppose that the partial schedule associated to the current node can be extended to a feasible schedule S_1 in which job j is scheduled somewhere after job k. We define a set $\mathcal{B} = U \backslash \{j, k\}$ of jobs. We also construct a schedule S'_1 by interchanging jobs j and k while the order of jobs belonging to $\mathcal{B}$ remains unchanged. Figure 7.4 illustrates schedules S_1 and S'_1.

To prove that interchanging jobs j and k does not increase the total weighted tardiness, we argue that the gain of interchanging jobs j and k, which is computed as $\mathrm{TWT}(S_1) - \mathrm{TWT}(S'_1)$, is greater than or equal to zero, no matter when job j is scheduled. Let $st_j(S)$ denote the start time of job j in schedule S. Remember that $st(\pi)$ denotes the start time of a sequence π. Let τ_1 be the difference between the start time of job j in S_1 and the start time of k in S'_1. If $st_k(S'_1)$ is less than $st_j(S_1)$ then τ_1

is negative, otherwise it is non-negative. By interchanging jobs j and k each job that belongs to set $\mathcal{B}$ may be shifted either to the right or to the left. Let $\tau_2 \geq 0$ be the maximum shift to the right of the jobs belonging to set $\mathcal{B}$. Notice that if all jobs in $\mathcal{B}$ are shifted to the left, then $\tau_2 = 0$. For each t as the start time of job j in S_1, Jouglet et al. (2004) define a function $\Gamma_{jk}(t, \tau_1, \tau_2)$ as follows:

$$\begin{aligned}\Gamma_{jk}(t, \tau_1, \tau_2) = & w_j \max\{0, t + p_j - d_j\} \\ & - w_k \max\{0, t + \tau_1 + p_k - d_k\} \\ & + w_k \max\{0, \hat{r}_k + p_k - d_k\} \\ & - w_j \max\{0, \hat{r}_j + p_j - d_j\} - \tau_2 \sum_{i \in \mathcal{B}} w_i.\end{aligned}$$

For the sub-problem of P where precedence and deadline constraints are removed, Jouglet et al. (2004) show that $\Gamma_{jk}(t, \tau_1, \tau_2)$ is a lower bound for the gain of interchanging jobs j and k when $t = st_j(S_1)$. This result can be improved by adding the gain of shifting the jobs which are tardy in both schedules S_1 and S_1'. We introduce the set $\mathcal{B}'$ of jobs where each job $i \in \mathcal{B}'$ is certainly a tardy job in S_1'. Let $\hat{\mathcal{P}}_i$ be the set of transitive predecessors of job i. The following set of jobs, which is a subset of $\mathcal{B}'$, is used in our implementations because the order based on which the jobs in $\mathcal{B}$ are scheduled has not yet been defined and therefore computing $\mathcal{B}'$ is not possible:

$$\left\{ i \in \mathcal{B} \middle| \hat{r}_j + p_j + \sum_{l \in (\mathcal{B} \cap \hat{\mathcal{P}}_i)} p_l + p_i \geq d_i \right\}.$$

Let $\tau_2' \geq 0$ be the minimum shift to the left of the jobs belonging to set $\mathcal{B}$. Note that at least one of the values τ_2' and τ_2 equals zero. We define the function $\hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau_2')$ as follows:

$$\hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau_2') = \Gamma_{jk}(t, \tau_1, \tau_2) + \tau_2' \sum_{i \in \mathcal{B}'} w_i.$$

The values τ_2 and τ_2' cannot be negative. Therefore, we immediately infer $\Gamma_{jk}(t, \tau_1, \tau_2) \leq \hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau_2')$. We need the following result:

Theorem 7.2. *$\hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau_2')$ is a valid lower bound for the gain of interchanging jobs j and k.*

Proof. If $\tau_2' = 0$, then $\Gamma_{jk}(t, \tau_1, \tau_2) = \hat{\Gamma}_{jk}(t, \tau_1, \tau_2, \tau_2')$ and the theorem holds based on Jouglet et al. (2004). If $\tau_2' > 0$, all jobs in $\mathcal{B}$ are shifted to the left at least τ_2' units. Also, τ_2 equals zero because no job is shifted to the right. For all jobs $i \in \mathcal{B}'$ we have $C_i(S_1) \geq C_i(S_1') \geq d_i$. Consequently, $\tau_2' \sum_{i \in \mathcal{B}'} w_i \geq 0$ is a lower bound for the gain of rescheduling jobs in $\mathcal{B}$. The value $w_j \max\{0, t+p_j-d_j\} - w_j \max\{0, \hat{r}_j+p_j-d_j\}$ equals the gain of rescheduling job j and the value $w_k \max\{0, \hat{r}_k + p_k - d_k\} - w_k \max\{0, t + \tau_1 + p_k - d_k\}$ equals the gain of rescheduling job k. By adding lower bounds for rescheduling gains of all jobs in $U = \mathcal{B} \cup \{j, k\}$, a lower bound for the gain of interchanging jobs j and k is obtained. □

In a general setting (problem P), however, job interchanges are not always feasible for every starting time t. We opt for verifying the feasibility of an interchange by ensuring that it does not cause any violation of deadlines and/or precedence constraints for all possible $t = st_j(S_1)$. Let Ψ be an upper bound for the completion time of the sequence S_1', computed as follows:

$$\Psi = \max\left\{\max\{\hat{r}_j + p_j, \hat{r}_k\} + p_k, \max_{i \in \mathcal{B}}\{\hat{r}_i\}\right\} + \sum_{i \in \mathcal{B}} p_i.$$

The following theorem provides the conditions under which for every possible $t = st_j(S_1)$ interchanging jobs j and k is feasible.

Theorem 7.3. *For each feasible schedule S_1, an alternative feasible schedule S_1' is created by interchanging jobs j and k, if the following conditions are satisfied:*

1. $\bar{\delta}_j - p_j \leq \bar{\delta}_k - \tau_1 - p_k$ *or* $\Psi \leq \hat{\delta}_k$;
2. $\tau_2 = 0$ *or* $\Psi \leq \min_{i \in \mathcal{B}}\{\hat{\delta}_i\}$.

Proof. We examine under which conditions the jobs belonging to the set $U = \mathcal{B} \cup \{j, k\}$ do not violate any of their deadlines and/or precedence constraints. Precedence constraints are not violated because jobs $j, k \in E_B$ are deliberately chosen such that $\mathcal{Q}_k \cap \mathcal{Q}_j = \mathcal{Q}_k$ and job j does not violate its deadline simply because $C_j(S_1') \leq C_j(S_1) \leq \bar{\delta}_j$.

Condition 1 ensures that job k does not violate its deadline. We argue that $t = st_j(S_1) \leq \bar{\delta}_j - p_j$. If $\bar{\delta}_j - p_j \leq \bar{\delta}_k - \tau_1 - p_k$ holds, then we infer $C_k(S_1') = t + \tau_1 + p_k \leq \bar{\delta}_k$. Also, if $\Psi \leq \hat{\delta}_k$, then all unscheduled jobs including j and k are completed at or before $\hat{\delta}_k$. Note that $\hat{\delta}_k$ is

preferred over $\bar{\delta}_k$ because $\bar{\delta}_k \leq \hat{\delta}_k$, thus condition 1 is less violated, and the inequality $\Psi \leq \hat{\delta}_k$ also implies $C_k(S'_1) \leq \bar{\delta}_k$.

Condition 2 verifies that no job in $\mathcal{B}$ violates its deadline. On the one hand, if $\tau_2 = 0$, then no job in $\mathcal{B}$ is shifted to the right, which means $C_i(S'_1) \leq C_i(S_1) \leq \bar{\delta}_i$ for each job $i \in \mathcal{B}$. On the other hand, if $\tau_2 > 0$ and $\Psi \leq \min_{i\in\mathcal{B}}\{\hat{\delta}_i\}$, then for all jobs $i \in \mathcal{B}$ we conclude: $C_i(S'_1) \leq \Psi \leq \min_{i\in\mathcal{B}}\{\hat{\delta}_i\} \leq \hat{\delta}_i$. Again, $\hat{\delta}_i$ is preferred over $\bar{\delta}_i$ because of the same reasoning as for the preference of $\hat{\delta}_k$ over $\bar{\delta}_k$. □

Jouglet et al. (2004) prove that if $w_j \geq w_k$ then the value $\Gamma_{jk}(\max\{d_j - p_j, \hat{r}_k + p_k\}, \tau_1, \tau_2)$ is the minimum gain obtained by interchanging jobs j and k for the setting where deadlines and precedence constraints are removed. We derive a more general result using the following lemma.

Lemma 7.1. *Let $f : t \to \alpha \max\{0, t-a\} - \beta \max\{0, t-b\} + C$ be a function defined on $[u, v]$ for $a, b, C \in \mathbb{R}$ and $\alpha, \beta, u, v \in \mathbb{R}_{\geq 0}$. The function f reaches a global minimum at value t^* computed as follows:*

$$t^*(\alpha, \beta, a, b, u, v) = \begin{cases} \min\{\bar{u}, v\} & \text{if } \alpha \geq \beta \\ u & \text{if } \alpha < \beta,\ b > a,\ \alpha(\bar{v} - \bar{u}) \geq \beta(\bar{v} - b) \\ v & \text{otherwise} \end{cases}$$

where $\bar{u} = \max\{u, a\}$ and $\bar{v} = \max\{v, b\}$.

Proof. Let f have a global minimum at value t^*. Depending on the values of the parameters α, β, a and b, the function f behaves differently. We discuss four possible cases for the parameter combinations to prove this lemma (see also Figure 7.5). In the two first cases, we assume that $\alpha \geq \beta$. Case (a): in this case, $a \leq b$, and then f is constant on interval $[u, a]$ and is non-decreasing on interval $[a, v]$, as shown in Figure 7.5(a). Case (b): $a > b$, f is constant on interval $[u, b]$, non-increasing on interval $[b, a]$ and non-decreasing on interval $[a, v]$, in line with Figure 7.5(b). The following results are valid for these two cases: 1- If $u \leq a \leq v$ then $t^* = a$. 2- If $a < u$, $t^* = u$ because f is always non-decreasing on interval $[u, v]$. 3- If $a > v$, $t^* = v$ because f is always non-increasing on interval $[u, v]$. We conclude that $t^* = \min\{\max\{a, u\}, v\}$ for the first two cases.

In the next two cases, we assume that $\alpha < \beta$. Case (c): $a < b$, f is constant for $[u, b]$, non-decreasing for $[b, a]$ and decreasing for $[a, v]$, as shown in Figure 7.5(c). In this case, t^* equals either u or v. On the one

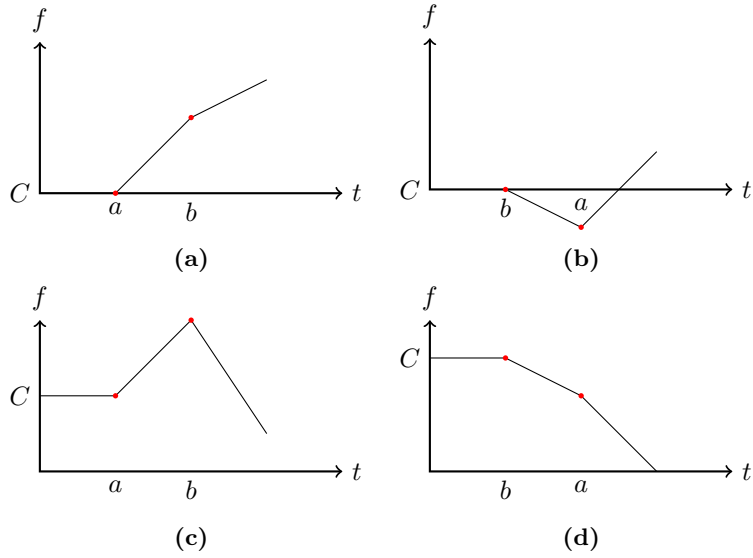


Figure 7.5: Four possible cases for the parameter combinations in the proof of Lemma 7.1.

hand, if $\alpha(b-\max\{a,u\}) \geq (\beta-\alpha)(\max\{v,b\}-b) \Leftrightarrow \alpha(\bar{v}-\bar{u}) \geq \beta(\bar{v}-b)$ then $f(v) \geq f(u)$ is inferred and $t^* = u$ is concluded. On the other hand, if $\alpha(\bar{v}-\bar{u}) < \beta(\bar{v}-b)$ then $t^* = v$ is concluded. Case (d): $a \geq b$, f is constant for $[u,b]$ and decreasing for $[b,v]$; see Figure 7.5(d). We find that t^* equals v for this case.

□

Corollary 7.1 below follows from Theorem 7.2, Theorem 7.3 and Lemma 7.1, if we choose $\alpha = w_j$, $\beta = w_k$, $a = d_j - p_j$, $b = d_k - \tau_1 - p_k$, $u = \hat{r}_k + p_k$, $v = \delta_j - p_j$ and $C = w_k \max\{0, \hat{r}_k + p_k - d_k\} - w_j \max\{0, \hat{r}_j + p_j - d_j\} - \tau_2 \sum_{i\in\mathcal{B}} w_i + \tau_2' \sum_{i\in\mathcal{B}'} w_i$. Let st_j^* be computed as follows:

$$st_j^* = t^*(w_j, w_k, d_j - p_j, d_k - \tau_1 - p_k, \hat{r}_k + p_k, \delta_j - p_j).$$

Corollary 7.1. *$\Gamma_{jk}^*(\tau_1,\tau_2,\tau_2') = \hat{\Gamma}_{jk}(st_j^*,\tau_1,\tau_2,\tau_2')$ is the minimum gain obtained by interchanging jobs j and k, provided that for every possible $st_j(S_1)$ interchanging jobs j and k is feasible.*

Case	$(\hat{r}_j + p_j - \hat{r}_k - p_k)$	$(p_j - p_k)$	$(\max_{i \in U}\{\hat{r}_i\} - \hat{r}_k - p_k)$	$(\hat{r}_j - \hat{r}_k)$	$(\max_{i \in U}\{\hat{r}_i\} - \hat{r}_k - p_j)$
1	$\leqslant 0$	< 0	$\geqslant 0$	-	-
2	$\leqslant 0$	< 0	< 0	$\leqslant 0$	> 0
3	$\leqslant 0$	< 0	< 0	$\leqslant 0$	$\leqslant 0$
4	$\leqslant 0$	< 0	< 0	> 0	-
5	$\leqslant 0$	$\geqslant 0$	$\geqslant 0$	-	-
6	$\leqslant 0$	$\geqslant 0$	< 0	-	-
7	> 0	< 0	-	-	-
8	> 0	$\geqslant 0$	-	-	-

Table 7.1: Interchange cases.

To compute $\Gamma^*_{jk}(\tau_1, \tau_2, \tau_2')$, the values of τ_1, τ_2 and τ_2' must be known. We establish an exhaustive list of cases for which τ_1, τ_2 and τ_2' can be computed, which is summarized in Table 7.1. Given a particular case, the values τ_1, τ_2 and τ_2' are computed as follows:

$$\tau_1 = \begin{cases} 0 & \text{Cases 1,5} \\ \max_{i \in U}\{\hat{r}_i\} - \hat{r}_k - p_k & \text{Case 2} \\ \max\{\hat{r}_j + p_j, \max_{i \in \mathcal{B}}\{\hat{r}_i\}\} - \hat{r}_k - p_k & \text{Cases 3,4,6} \\ \hat{r}_j + p_j - \hat{r}_k - p_k & \text{Cases 7,8} \end{cases}$$

$$\tau_2 = \begin{cases} p_k - p_j & \text{Case 1} \\ \max_{i \in U}\{\hat{r}_i\} - \hat{r}_k - p_j & \text{Case 2} \\ 0 & \text{Cases 3,5,6} \\ \max\{\hat{r}_j + p_j, \max_{i \in \mathcal{B}}\{\hat{r}_i\}\} - \hat{r}_k - p_j & \text{Case 4} \\ \hat{r}_j - \hat{r}_k & \text{Case 7} \\ \hat{r}_j + p_j - \hat{r}_k - p_k & \text{Case 8} \end{cases}$$

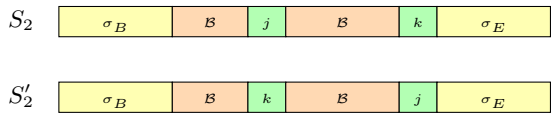


Figure 7.6: Schedules S_2 and S_2'.

$$\tau_2' = \begin{cases} 0 & \text{Cases 1,2,4,5,7,8} \\ \hat{r}_k - \hat{r}_j & \text{Case 3} \\ \hat{r}_k + p_k - \max\{\hat{r}_j + p_j, \max_{i\in\mathcal{B}}\{\hat{r}_i\}\} & \text{Case 6} \end{cases}$$

Following the above results, the first part of Dominance rule 7.4 is derived.

Dominance rule 7.4 ($DR_{7.4}$; first part)**.** *Given an FB node associated with* $(\sigma_B, \emptyset)$*, if there exist two non-identical jobs* $j, k \in E_B$ *with* $\mathcal{Q}_k \cap \mathcal{Q}_j = \mathcal{Q}_k$ *and the inequality* $\Gamma^*_{jk}(\tau_1, \tau_2, \tau_2') > 0$ *holds, then* $(\sigma_B|j, \emptyset)$ *dominates* $(\sigma_B|k, \emptyset)$*.*

7.2.2.2 Interchanging jobs in a BB node

Let $j, k \in E_E$ where jobs j and k are not identical. We also assume that any unscheduled predecessor of job k is also a predecessor of job j. In other words, we have $\mathcal{P}_k \cap \mathcal{P}_j \cap U = \mathcal{P}_k \cap U$. Consider a BB node of the search tree with decision job k. The partial schedule associated with the current node can be extended to a feasible schedule S_2 in which job j is scheduled before job k but after all jobs in the sequence σ_B. The set $\mathcal{B}$ is the set of all remaining unscheduled jobs where $\mathcal{B} = U\backslash\{j, k\}$. Let schedule S_2' be constructed by interchanging jobs j and k while keeping the order based on which the jobs belonging to $\mathcal{B}$ will be scheduled. Figure 7.6 illustrates schedules S_2 and S_2'.

For each t as the start time of job j in S_2, we define a function $\Delta_{jk}(t)$ as follows:

$$\begin{aligned} \Delta_{jk}(t) = {} & w_j \max\{0, t + p_j - d_j\} \\ & - w_k \max\{0, t + p_k - d_k\} \\ & + w_k \max\{0, st(\sigma_E) - d_k\} \\ & - w_j \max\{0, st(\sigma_E) - d_j\} - \max\{0, p_k - p_j\} \sum_{i\in\mathcal{B}} w_i. \end{aligned}$$

In a BB node, for each t as the start time of job j, $\Delta_{jk}(t)$ is a lower bound of the gain of interchanging jobs k and j, if the conditions of Theorem 7.4 are satisfied. Theorem 7.4 provides the conditions on which for every possible $t = st_j(S_1)$ interchanging jobs j and k is feasible.

Theorem 7.4. *For each feasible schedule S_2, a feasible schedule S_2' can be created by interchanging jobs j and k, if the following conditions are satisfied:*

1. $st(\sigma_E) \leq \hat{\delta}_j$;
2. $p_k - p_j \leq 0$ *or* $st(\sigma_E) - p_j \leq \min_{i \in \mathcal{B}} \hat{\delta}_i$.

Proof. Similar to the proof of Theorem 7.3. □

Corollary 7.2 follows from Theorem 7.4 and Lemma 7.1, if we choose $\alpha = w_j$, $\beta = w_k$, $a = d_j - p_j$, $b = d_k - p_k$, $u = C_{\max}(\sigma_B)$, $v = st(\sigma_E) - p_k - p_j$ and $C = w_k \max\{0, st(\sigma_E) - d_k\} - w_j \max\{0, st(\sigma_E) - d_j\} - \max\{0, p_k - p_j\} \sum_{i \in \mathcal{B}} w_i$. Let $st_j^{*\prime}$ be computed as follows:

$$st_j^{*\prime} = t^*(w_j, w_k, d_j - p_j, d_k - p_k, C_{\max}(\sigma_B) + \sum_{i \in \mathcal{P}_j \cap U} p_i, st(\sigma_E) - p_k - p_j).$$

Corollary 7.2. *$\Delta_{jk}^* = \Delta_{jk}(st_j^{*\prime})$ is the minimum gain obtained by interchanging jobs j and k, provided that for every possible $t = st_j(S_1)$ interchanging jobs j and k is feasible.*

Following the above results, the second part of Dominance rule 7.4 is derived.

Dominance rule 7.4 ($DR_{7.4}$; second part)**.** *Given a BB node associated with (σ_B, σ_E), if there exist two non-identical jobs $j, k \in E_E$ with $\mathcal{P}_k \cap \mathcal{P}_j \cap U = \mathcal{P}_k \cap U$ and $\Delta_{jk}^* > 0$, then $(\sigma_B, j|\sigma_E)$ dominates $(\sigma_B, k|\sigma_E)$.*

7.2.3 Dominance rule based on job insertion

We describe a dominance rule based on job insertion. This dominance rule, similar to the dominance rule based on job interchange, consists of two parts. The first part deals with the insertion of a job in an FB node whereas the second part deals with the insertion of a job in a BB node.

Figure 7.7: Schedule S_1''.

7.2.3.1 Inserting a job in an FB node

In an FB node, let $j, k \in E_B$ where jobs j and k are not identical. Again we assume that $\hat{r}_k < \hat{r}_j + p_j$ and $\hat{r}_j < \hat{r}_k + p_k$, otherwise Dominance rule 7.2 enforces scheduling the job with smaller $\hat{r}$ before the job with larger $\hat{r}$ (remind that $\hat{r}_j = \bar{r}_j$ and $\hat{r}_k = \bar{r}_k$ because all predecessors of jobs j and k have already been scheduled and therefore the branching decisions cover precedence constraints propagation). Consider an FB node of the search tree in which job k is scheduled after the jobs in sequence σ_B. Assume that the partial schedule associated with the current node can be extended to the feasible schedule S_1 depicted in Figure 7.4. We construct a schedule S_1'' by inserting the job j before job k while keeping the order of jobs belonging to $\mathcal{B}$. Figure 7.7 illustrates the construction of the schedule S_1''.

Let τ_3 be the maximum shift to the right of the jobs belonging to $\mathcal{B}$, which is computed as follows:

$$\tau_3 = \max\left\{0, \hat{r}_j + p_j + p_k - \max\left\{\hat{r}_k + p_k, \min_{i\in\mathcal{B}}\{\bar{r}_i\}\right\}\right\}.$$

For each t as the start time of job j in schedule S_1, we define a function $\Gamma'_{jk}(t, \tau_3)$ as follows:

$$\begin{aligned}\Gamma'_{jk}(t,\tau_3) = {} & w_j \max\{0, t + p_j - d_j\} \\ & - w_k \max\{0, \hat{r}_j + p_j + p_k - d_k\} \\ & + w_k \max\{0, \hat{r}_k + p_k - d_k\} \\ & - w_j \max\{0, \hat{r}_j + p_j - d_j\} - \tau_3 \sum_{i\in OJ} w_i.\end{aligned}$$

Job insertion, similar to job interchange, is not always feasible for every starting time t of job j. We verify feasibility of an insertion by ensuring that it does not cause any deadline and/or precedence-constraint violation for all possible $t = st_j(S_1)$. Let Ψ' be an upper bound for the completion

time of the sequence S'_1, computed as follows:

$$\Psi' = \max\left\{\hat{r}_j + p_j + p_k, \max_{i\in\mathcal{B}}\{\hat{r}_i\}\right\} + \sum_{i\in\mathcal{B}} p_i.$$

The following theorem provides the conditions under which for every possible $t = st_j(S_1)$ inserting job j before job k is feasible.

Theorem 7.5. *For each feasible schedule S_1, another feasible schedule S''_1 can be created by inserting job j before job k if the following conditions hold:*

1. $\hat{r}_j + p_j + p_k \leq \hat{\delta}_k$;
2. $\tau_3 = 0$ *or* $\Psi' \leq \min_{i\in\mathcal{B}}\{\hat{\delta}_i\}$.

Proof. Similar to the proof of Theorem 7.3. □

Corollary 7.3 below follows from Theorem 7.5.

Corollary 7.3. $\Gamma'^{*}_{jk}(\tau_3) = \Gamma'_{jk}(\hat{r}_k + p_k, \tau_3) = \Gamma_{jk}(\hat{r}_k + p_k, \hat{r}_j + p_j - \hat{r}_k - p_k, \tau_3)$ *is the minimum gain obtained by inserting job j before job k provided that for every possible $t = st_j(S_1)$ inserting job j before job k is feasible.*

Following the above results, the first part of Dominance rule 7.5 is derived.

Dominance rule 7.5 ($DR_{7.5}$; first part)**.** *Consider an FB node associated with $(\sigma_B, \emptyset)$. If there exist two non-identical jobs $j, k \in E_B$ for which the inequality $\Gamma'^{*}_{jk}(\tau_3) > 0$ holds, then $(\sigma_B|j, \emptyset)$ dominates $(\sigma_B|k, \emptyset)$.*

7.2.3.2 Inserting a job in a BB node

In a BB node, let $j, k \in E_E$ where jobs j and k are not identical. Consider a node of the search tree in which job k is scheduled before sequence σ_E. Assume that the partial schedule associated with the current node can be extended to the feasible schedule S_2 depicted in Figure 7.6. We also construct a schedule S''_2 by inserting the job j to be scheduled after job k but before the jobs in the sequence σ_E and by keeping the order of jobs belonging to $\mathcal{B}$. Figure 7.8 illustrates schedule S''_2.

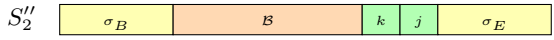


Figure 7.8: Schedule S_2''.

For each t, which is the start time of job j in schedule S_2, we define the function $\Delta'_{jk}(t)$ as follows:

$$\begin{aligned}\Delta'_{jk}(t) = {} & w_j \max\{0, t + p_j - d_j\} \\ & - w_k \max\{0, st(\sigma_E) - p_j - d_k\} \\ & + w_k \max\{0, st(\sigma_E) - d_k\} \\ & - w_j \max\{0, st(\sigma_E) - d_j\}.\end{aligned}$$

Similarly to the previous results, for each feasible schedule S_2, a feasible schedule S_2'' is constructed by inserting jobs j after job k, if $st(\sigma_E) \leq \hat{\delta}_j$. The following corollary is obtained:

Corollary 7.4. *$\Delta'^{*}_{jk} = \Delta'_{jk}(C_{\max}(\sigma_B) + \sum_{i \in \mathcal{P}_j \cap U} p_i)$ is the minimum gain obtained by inserting job j after job k provided that $st(\sigma_E) \leq \hat{\delta}_j$.*

Following the above results, the second part of Dominance rule 7.5 is derived.

Dominance rule 7.5 ($\mathrm{DR}_{7.5}$; second part)**.** *Consider a BB node associated with (σ_B, σ_E). If there exist two non-identical jobs $j, k \in E_E$ for which the inequality $\Delta'^{*}_{jk} > 0$ holds, then $(\sigma_B, j|\sigma_E)$ dominates $(\sigma_B, k|\sigma_E)$.*

7.2.4 Dominance rules on scheduled jobs

The *dominance theorem of dynamic programming* (see Jouglet et al., 2004) is another existing theorem that can be used to eliminate nodes in the search tree. It compares two partial sequences that contain identical subsets of jobs and eliminates the one having the larger total weighted tardiness. When total weighted tardiness values are the same, then only one of the sequences is kept. Let us consider two feasible partial sequences σ_1 and σ_2 (σ_2 is a feasible permutation of σ_1) of k jobs, where $k < n$. Let $\mathcal{C}$ be the set of jobs in either σ_1 or σ_2. We are going to decide whether it is advantageous to replace σ_2 by σ_1 in all (partial) schedules in which σ_2 orders the last k jobs. The set of scheduled jobs and the set of unscheduled

jobs are identical for both σ_1 and σ_2. Sequence σ_1 is as good as sequence σ_2 if it fulfills one of the following conditions:

1. $C_{\max}(\sigma_1) \leq C_{\max}(\sigma_2)$ and $\mathrm{TWT}(\sigma_1) \leq \mathrm{TWT}(\sigma_2)$;
2. $C_{\max}(\sigma_1) > C_{\max}(\sigma_2)$ and the following inequality also holds:

$$\mathrm{TWT}(\sigma_1) + (\min_{i \in U}\{\bar{r}_i^{\sigma_1}\} - \min_{i \in U}\{\bar{r}_i^{\sigma_2}\}) \sum_{i \in U} w_i \leq \mathrm{TWT}(\sigma_2),$$

where $\bar{r}_i^{\sigma_1}$ is the updated release date associated with the sequence σ_1 and $\bar{r}_i^{\sigma_2}$ is the updated release date associated with the sequence σ_2.

Jouglet et al. (2004) determine the sequences that can be replaced by a dominant permutation. They find that sequence σ_1 dominates sequence σ_2 if the following two conditions hold:

1. sequence σ_1 is as good as sequence σ_2;
2. sequence σ_2 is not as good as σ_1 or σ_1 has lexicographically smaller release dates than σ_2.

Note that the second condition enforces a tie-breaking rule where a lexicographical number associated to each sequence is computed and among those sequences that are equivalent, the one with lower lexicographic number is selected. To avoid conflicts with Dominance rule 7.2, jobs are renumbered in non-decreasing order of their release dates r_j.

Dominance rule 7.6 ($\mathrm{DR}_{7.6}$)**.** *If there exists a better feasible permutation of σ_B and/or a better feasible permutation of σ_E, then the node (σ_B, σ_E) is fathomed.*

If $\sigma_E = \emptyset$ and there is a better feasible permutation of σ_B, then the dominance is proven similarly to Theorem 13.6 in Jouglet et al. (2004). If $\sigma_E \neq \emptyset$, then all jobs belonging to the set U will be scheduled between $C_{\max}(\sigma_B)$ and $st(\sigma_E) = C_{\max}(\sigma_B) + \sum_{j \in U} p_j$. Therefore, all permutations of σ_E start at time $st(\sigma_E)$ and if there exists at least one better feasible permutation of σ_E, then fathoming the node associated with (σ_B, σ_E) does not eliminate the optimal solution.

Dominance rule 7.6 where only permutations of the last k jobs are considered, is referred to as $\mathrm{DR}_{7.6}^k$. Computing $\mathrm{DR}_{7.6}^n$ amounts to enumerating all $\mathrm{O}(n!)$ feasible solutions, which would yield an optimal solution

k	n			
	20	30	40	50
2	0.0043	-	-	-
3	**0.0038**	0.045	-	-
4	0.0039	**0.039**	5.157 (1)	-
5	0.0042	0.039	3.499	15.895 (15)
6	0.0043	0.041	**3.130**	**13.358 (13)**
7	0.0050	0.046	4.741 (1)	14.301 (15)
8	-	0.092	7.459 (1)	22.470 (17)

Table 7.2: Average CPU times (in seconds; first number) and number of unsolved instances within the time limit (between brackets, if any; out of 864) for different choices of k in BB1 run on Ins.

but is computationally prohibitive. In our B&B algorithm, we therefore choose $k < n$. There is a trade-off between the computational effort needed to compute $\mathrm{DR}_{7.6}^{k}$ and the improvement achieved by eliminating dominated nodes. Based on initial experiments (see Table 7.2; more details on the instance generation are provided in Section 5.4), we observe that the algorithms perform worse when $k > 6$. We also notice that it is not efficient to use $\mathrm{DR}_{7.6}^{k}$ when $k > |U|$ because the computational effort to solve the subproblem consisting of the remaining $|U|$ jobs is less than the computational effort needed to enumerate all feasible permutations of the last k jobs. Thus, $k = \min\{|\sigma_B|, |U|, 6\}$ while scheduling forward and $k = \min\{|\sigma_E|, |U|, 6\}$ when scheduling backward.

We observe that in BB2 with unequal release dates, at certain moments during the search procedure, we switch from forward to backward branching, which forces us to start with $k = |\sigma_E| = 0$ and we thus lose a number of pruning opportunities.

7.3 Initial upper bound

Although for most of the instances the B&B algorithm finds a reasonably good solution (a tight upper bound) quickly, there are instances for which feasible solutions are encountered only after a large part of the search tree has been scanned. Therefore, we initialize the upper bound in the root node of the B&B algorithm using a stand-alone (heuristic) procedure, which we refer to as *time-window heuristic* (TWH).

Algorithm 7.1 Time-window heuristic (TWH)

Input: a sequence σ

for $itr = 1$ to 2 **do**
  IMPROVE_BY_SWAP
  $k = 0$
  while $k \leq 50$ **do**
    if k even **then**
      $start = 0$
    else
      $start = \min\{minsize, maxsize/2\}$
    while $start + minsize \leq n$ **do**
      $end = \min\{start + maxsize, n\}$
      SP $\leftarrow$ CONST_SP($\sigma, start, end$)
      $\sigma_{\text{SP}} \leftarrow$ SOLVE_BB(SP)
      $\sigma' \leftarrow$ COPYSEQ($\sigma, \sigma_{\text{SP}}, start, end$)
      if TWT(σ') $<$ TWT(σ) **then**
        $\sigma \leftarrow \sigma'$
    $k = k + 1$

Output: TWT(σ)

The key idea of our TWH is to iteratively locally improve a given sequence of jobs within a varying time window (Algorithm 7.1); similar ideas have already been proposed in the literature (Debels and Vanhoucke, 2007; Kinable et al., 2014). It starts with any given sequence (note that finding a feasible sequence might be very difficult for some instances, so we also allow infeasible sequences). Then to locally improve the solution, the algorithm constructs a number of subproblems. Each subproblem is defined by two positions: a *start* position and an *end* position. The subproblem tries to optimally resequence the jobs that are positioned between the given *start* and *end* positions in the initial sequence such that the completion time of the subsequence does not exceed the start time of the job in the position $end + 1$. This additional condition is fulfilled by updating the deadline of all jobs j in the subproblem to $\delta_j^{SP} = \min\{\delta_j, st_{end+1}(\sigma)\}$ and updating the release date of all jobs j in the subproblem to $r_j^{SP} = \max\{r_j, C_{start-1}(\sigma)\}$.

In TWH, the subprocedure IMPROVE_BY_SWAP is a naive local search procedure in which each pair of jobs is examined for swapping exactly once, in a steepest descent fashion. The length of the subse-

Instance Set		total	feas	TWH		
				fnd	opt	gap
Ins^{L}	$n = 30$	432	401	397	300	0.012
	$n = 40$	432	395	392	313	0.011
	$n = 50$	432	397	395	302	0.025
Ins^{PAN}	$n = 30$	100	100	100	59	0.003
	$n = 40$	100	100	100	72	0.001
	$n = 50$	94*	94	89	50	0.001
Ins^{TAN}	$n = 40$	875	875	875	542	0.019
	$n = 50$	875	875	875	545	0.015

$*$ the optimal solutions are only available for 94 instances.

Table 7.3: The performance of TWH

quence to be reoptimized is in between $minsize = 10$ and $maxsize = \min\{\max\{n/2, 10\}, 20\}$. Given a *start* and an *end* position, CONST_SP constructs the associated subproblem and SOLVE_BB solves the subproblem using the same branch-and-bound algorithms explained in this chapter. A new sequence is constructed using COPYSEQ.

The input sequence for TWH is the result of a dynamic priority rule that stepwise schedules jobs (from time 0 onwards) that are eligible according to the precedence constraints (meaning that all predecessors were previously already selected) and whose release date has already been reached; if no job is eligible in this way, then the algorithm proceeds to the earliest ready time of all jobs for which all predecessors have already been scheduled. If multiple jobs are eligible, then priority is given to the one with the earliest deadline and the lowest processing time. In the computation of the eligible set of jobs the deadline constraints are ignored. Therefore, the resulting sequence might not be feasible to P. In such cases, we add a large infeasibility cost to the objective function in the hope of finding a feasible solution during TWH.

The upper bound that is the output of TWH improves the runtime for those instances for which the branch-and-bound algorithm fails to find a feasible solution fast. Furthermore, this upper bound turns out to be optimal for most of the instances of P and for those instances for which the optimal solution is not found, the optimality gap is very low; see Table 7.3 (see Section 5.4 for more details on the instances in Ins^{L} and Section 7.4.3 for more details on the instances in Ins^{PAN} and Ins^{TAN}). To evaluate the efficiency of TWH, we have run some computational ex-

Ins^{L}			Ins^{PAN}	Ins^{TAN}	
$n = 30$	$n = 40$	$n = 50$		$n = 40$	$n = 50$
0.04	0.09	0.17	0.08	0.31	0.66

Table 7.4: Average CPU times (in seconds) of upper bound computation for different instance sets

periments, the results of which are reported in Table 7.3 and Table 7.4. In Table 7.3, column *total* contains the total number of instances and the values in column *feas* represent the number of instances for which at least one feasible solution exists. Column *fnd* reports the number of times TWH finds a feasible solution, column *opt* counts the number of optimal solutions found and column *gap* states the average optimality gap, averaged only for the instances for which the optimal solution was not found by TWH. The average optimality gap is computed as the average value of ((UB − OPT)/UB), with UB the output of TWH. Table 7.4 reports the average CPU times for the same subset of instances studied in Table 7.3. Note that the column that reports the average CPU times for Ins^{PAN} pertains to all instances with $n = 30, 40$ and 50.

7.4 Computational results

All algorithms have been implemented in VC++ 2010. All computational results were obtained on a laptop Dell Latitude with 2.6 GHz Core(TM) i7-3720QM processor, 8GB of RAM, running under Windows 7. In all our experiments, the time limit is set to 1200 seconds. If an instance is not solved to guaranteed optimality, it is said to be 'unsolved' for the procedure. Throughout this section, we report averages computed only over the solved instances.

Since $\text{LB}_2^{\text{SS}_\delta}$ and $\text{LB}_2^{\text{SS}_\text{r}}$ require the same computational effort, we decide not to use $\text{LB}_2^{\text{SS}_\text{r}}$ in our B&B algorithms, where enumeration of child nodes is more efficient than extra computation of a weak bound. also, since $\text{LB}_3^{\text{SS}_\text{r}}$ and $\text{LB}_3^{\text{SS}_\delta}$ are equally expensive in terms of computational effort, we decide not to use $\text{LB}_3^{\text{SS}_\text{r}}$.

In our final implementation, we will not compute all the bounds for all the nodes because this consumes too much effort. We start with computing LB_T, LB_0 and LB_{SV1} for the unscheduled jobs. Let S_{best} be the best feasible schedule found. If the node is fathomed by $\text{DR}_{7.1}$, then we back-

Scenario	$DR_{7.2}$	$DR_{7.3}$	$DR^2_{7.6}$	$DR_{7.4}$	$DR_{7.5}$	$DR^k_{7.6}$	$DR_{7.1}$
1	✓	✓	✓	-	-	-	-
2	✓	✓	✓	✓	-	-	-
3	✓	✓	✓	✓	✓	-	-
4	✓	✓	✓	✓	✓	✓	-
5	✓	✓	✓	-	✓	✓	✓
6	✓	✓	✓	✓	-	✓	✓
7	✓	✓	-	✓	✓	-	✓
8	✓	✓	✓	✓	✓	✓	✓

Table 7.5: The list of scenarios.

track; otherwise if $\mathrm{TWT}(S_P) + \lceil \mathrm{LB}_0 + \mathrm{LB}_{\mathrm{SV1}} \rceil \times 1.4 < \mathrm{TWT}(S_{best})$ then we do not compute the remaining lower bounds and continue branching. If the latter equality does not hold, then we anticipate that with a better bound we might still be able to fathom the node, and we compute LB_3 and/or $\mathrm{LB}_3^{\mathrm{SS}_\delta}$. For all lower bounds we choose $k_{\max} = 0$ if $OS < 0.5$ and $k_{\max} = 1$ otherwise.

7.4.1 Dominance rules

In each node of the B&B algorithm, dominance rules are tested. Based on some preliminary experiments, we find that applying the rules in the following order performs well, and we will therefore follow this order throughout the algorithm:

$$\mathrm{DR}_{7.2}, \mathrm{DR}_{7.3}, \mathrm{DR}^2_{7.6}, \mathrm{DR}_{7.4}, \mathrm{DR}_{7.5}, \mathrm{DR}^k_{7.6}, \mathrm{DR}_{7.1}.$$

In order to evaluate the effectiveness of the rules, we examine a number of scenarios with respect to the selection of the implemented bounds; the list of scenarios is given in Table 7.5. Scenario 1 includes the simplest combination of dominance rules, namely $\mathrm{DR}_{7.2}$, $\mathrm{DR}_{7.3}$ and $\mathrm{DR}^2_{7.6}$. From Scenario 2 to Scenario 4, extra rules are gradually added. In Scenario 5, all dominance rules are active except $\mathrm{DR}_{7.4}$, and in Scenario 6, only $\mathrm{DR}_{7.5}$ is inactive. Scenario 7 similarly includes all dominance rules except $\mathrm{DR}_{7.6}$. Finally, in Scenario 8, all dominance rules are active.

For each of these implementations, we report the average CPU times and the average number of nodes explored in the search tree in Table 7.6; the results pertain to the instances of Ins with $n = 20, 30$. Scenarios 2 and 3 show the effect of $\mathrm{DR}_{7.4}$ and $\mathrm{DR}_{7.5}$. In Scenario 2, $\mathrm{DR}_{7.4}$ improves

Method	Scenario	$n = 20$ CPU	$n = 20$ Nodes	$n = 30$ CPU	$n = 30$ Nodes
BB1	1	0.004	12521	-	-
	2	0.003	4310	2.956	4388157
	3	0.003	4279	3.547	4382728
	4	0.004	839	0.260	48410
	5	0.008	1912	0.075	10095
	6	0.003	488	0.016	3442
	7	0.772	1044361	-	-
	8	0.003	487	0.039	3451
BB2	1	0.003	11698	-	-
	2	0.002	3609	1.506	2182869
	3	0.002	3271	1.482	1669982
	4	0.003	980	0.260	91985
	5	0.004	1658	0.135	30474
	6	0.002	490	0.055	9750
	7	0.155	239389	-	-
	8	0.002	427	0.047	7464

'-' means that the implementation fails to solve many instances within the time limit.

Table 7.6: The effect of the dominance rules.

the performance of both algorithms whereas in Scenario 3, $DR_{7.5}$ has a beneficial effect only for BB2. Scenario 4 reflects the impact of $DR_{7.6}$ for k jobs.

Comparing Scenario 5 to Scenario 8, we see that inclusion of $DR_{7.1}$ has a strong beneficial effect on both algorithms; the effect is strongest in BB2 because tighter bounds can be computed by scheduling backward. From Table 7.6, we learn that apart from $DR_{7.2}$, which is always crucial in total tardiness scheduling problems, the most important dominance rule is $DR_{7.6}$: deactivating this rule triggers a huge increase in the average number of nodes and the average CPU times; incorporating $DR_{7.4}$ also has a marked effect (compare Scenarios 5 and 8). Among all dominance rules tested, $DR_{7.5}$ is the least important; removing $DR_{7.5}$ slightly increases the node count and the runtimes in BB2. In BB1, removing $DR_{7.5}$ even decreases the number of nodes and the runtimes; it turns out that for $n > 30$, however, the effect of $DR_{7.5}$ is also (slightly) beneficial for BB1, and so we decide to adopt Scenario 8 as the final setting in which the experiments in the following sections will be run.

α	Method	n		
		10	20	30
10	ASF	0.81	–	–
	ASF′	0.80	–	–
	TIF	0.43	2.02	53.47 (3)
	TIF′	0.64	2.97	88.17 (12)
	BB1	0.00	0.00	0.02
	BB2	0.00	0.00	0.03
100	ASF	0.92	–	–
	ASF′	0.95	–	–
	TIF	6.54	–	–
	TIF′	21.78	–	–
	BB1	0.00	0.00	0.06
	BB2	0.00	0.00	0.06

Table 7.7: Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 432) for the MIP formulations and the B&B algorithms run on Ins with $n = 10$, 20 and 30.

As a side note, we observe that for all the foregoing dominance rules, after the root node, omitting the precedence constraints implied by sets $\mathcal{Q}_j$ and $\mathcal{P}_j$ from the updates of $\bar{r}_j$ and $\bar{\delta}_j$ has only little effect. We will therefore not include these precedence constraints into the updated release dates and deadlines and thus avoid the additional computational overhead.

7.4.2 Branch-and-bound algorithms

In this section we discuss the performance of our B&B algorithms. In Table 7.7 we compare the performance of BB1 and BB2 with the MIP formulations discussed in Section 5.3. In this table as well as in the following, we report the average runtime and the number of unsolved instances (if there are any).

Based on Table 7.7, the B&B algorithms BB1 and BB2 both clearly outperform the MIP formulations regardless of the size of the processing times. Table 7.8 shows the performance of BB1 and BB2 applied to the larger instances of Ins ($n = 40$ and 50) that cannot be solved by the MIP formulations. On average, BB1 performs better than BB2, although this

α	Method	n	
		40	50
10	BB1	1.26	16.99 (1)
	BB2	1.65	35.73 (6)
100	BB1	5.00	14.00 (12)
	BB2	3.38 (1)	18.28 (12)

Table 7.8: Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 432) for BB1 and BB2 run on Ins with $n = 40$ and 50.

Method	n	OS			
		0	0.25	0.50	0.75
BB1	30	0.08	0.04	0.02	0.01
	40	11.85	0.55	0.10	0.02
	50	50.81 (13)	12.63	0.68	0.06
BB2	30	0.05	0.10	0.03	0.01
	40	7.37 (1)	2.37	0.32	0.03
	50	44.17 (12)	57.12 (6)	8.66	0.10

Table 7.9: Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 216) for different choices of n and OS in BB1 and BB2 run on Ins.

does not hold for all parameter settings (more details follow below). BB1 solves all instances with 40 jobs and fails to solve around 1.5% of the instances with 50 jobs. BB2 fails to solve one instance with 40 jobs and around 2% of the instances with 50 jobs. We will indicate below that all these unsolved instances belong to a specific class; it is worth mentioning that the difficult instances are not the same for the two B&B algorithms.

The number of precedence constraints obviously affects the performance of the algorithms. On the one hand, by adding precedence constraints, the set of feasible sequences shrinks; on the other hand, the lower bounds also become less tight. The net result of these two effects is a priori not predictable. For instance classes without release dates and deadlines ($r_j = 0$ and $\delta_j = \infty$), the quality of the lower bound is very good when $OS = 0$, therefore the effect of a weaker bound due to higher OS will be more pronounced than when release dates and deadlines are also imposed.

To identify the classes of difficult instances, we focus on case $n = 50$. Table 7.10 shows the outcomes of the experiments for each combination

Method	τ	ρ	OS			
			0	0.25	0.50	0.75
BB1	0	0.05	0.27	0.50	0.18	0.03
		0.25	26.84	11.89	0.51	0.05
		0.50	127.59	25.92	1.96	0.06
	0.5	0.05	1.60	5.16	0.44	0.05
		0.25	35.16 (2)	8.86	0.72	0.07
		0.50	439.09 (11)	57.55	1.86	0.09
	1	0.05	1.77	0.44	0.15	0.04
		0.25	1.14	0.50	0.16	0.05
		0.50	0.49	2.86	0.16	0.06
BB2	0	0.05	0.21	1.01	0.25	0.03
		0.25	0.27	2.79	0.36	0.05
		0.50	0.49	3.11	0.37	0.06
	0.5	0.05	1.66	76.47	5.97	0.12
		0.25	77.95 (1)	158.78	17.96	0.20
		0.50	544.25 (11)	338.02 (6)	52.42	0.30
	1	0.05	1.75	0.43	0.15	0.06
		0.25	1.09	0.51	0.29	0.05
		0.50	0.47	3.15	0.19	0.05

Table 7.10: Average CPU times (in seconds) and number of unsolved instances within the time limit (out of 24) for different choices of τ, ρ and OS in BB1 and BB2 run on Ins with $n = 50$.

of τ, ρ and OS. According to this table, the most time-consuming class of instances is the one where release dates are neither loose nor tight ($\tau = 0.50$), due dates are loose ($\rho = 0.50$) and the set of precedence constraints is empty ($OS = 0$). No clear pattern can be distinguished for the algorithmic performance as a function of the tightness of the deadlines, so these results are excluded from the table. The unsolved instances are distributed differently for the two algorithms, although $\tau = 0.5$ in all and $OS = 0$ in most of the unsolved instances. For example, BB1 solves all instances with $OS = 0.25$ whereas BB2 does not solve six of these instances. Also, BB1 fails to solve two instances with $\rho = 0.25$ whereas this occurs for only one instance with $\rho = 0.25$ for BB2.

We will represent each class of instances by a triple (τ, ρ, OS). As mentioned before, the hardest class of instances for both algorithms is

$(0.5, 0.5, 0)$. The class $(0, 0.5, 0)$, which seems to be the third most difficult class for BB1, is very easy for BB2. Also, $(0.5, 0.5, 0.25)$, which is the second hardest class for BB2, does not require very high runtimes from BB1. We infer that BB2 is better than BB1 when release dates are equal (zero); in this case (cf. Section 7.1) stronger bounds are computed in backward scheduling. Conversely, BB1 is better than BB2 when release dates are not equal (especially when $\tau = 0.50$). With unequal release dates, backward branching cannot start from the root node but rather only after a certain number of jobs have already been scheduled. Because branching forward increases the earliest possible starting and completion times of jobs, the trivial lower bound and the Lagrangian-based lower bounds will be stronger for BB1 than those for BB2. As explained at the end of Section 7.2.4 and contrary to BB2, in BB1 k is never restarted in the computation of $\mathrm{DR}^k_{7.6}$ and therefore we do not lose any pruning opportunity.

7.4.3 Experiments for subproblems of P

In this section, we present the results of our B&B algorithms for subproblems of P that have also been studied in earlier literature. Two sets of benchmark instances for subproblems of P are also used in our experiments; these are referred to as $\mathrm{Ins}^{\mathrm{TAN}}$ and $\mathrm{Ins}^{\mathrm{PAN}}$ and are discussed in Sections 7.4.3.1 and 7.4.3.2, respectively.

7.4.3.1 A single-machine problem with precedence constraints: $1|prec|\sum w_j T_j$

One special case of P is single machine scheduling with precedence constraints where the objective is to minimize the total weighted tardiness. From our observations in Section 7.4.2, we know that we only need to consider BB2 for this subproblem because all release dates are zero, and so we compare the performance of BB2 with the SSDP algorithm proposed by Tanaka and Sato (2013). We apply both algorithms to the benchmark instances $\mathrm{Ins}^{\mathrm{TAN}}$ obtained from Tanaka and Sato (2013). For these instances, parameter Pr denotes the probability that each arc $(i, j) \in N \times N$ with $i \neq j$ is present in the precedence graph. Note that the resulting precedence graph may contain transitive arcs. In such cases, the transitive reduction is computed and used as input to BB2. Table 7.11 shows the computational results for our B&B algorithms and for the SSDP algorithm (which was run on the same computer). SSDP solves instances

Pr	$n = 40$		$n = 50$	
	BB2	SSDP	BB2	SSDP
0	0.04	0.04	0.26	0.06
0.005	0.49	0.35	1.83	0.71
0.01	0.61	0.51	4.98	2.05
0.02	0.80	1.67	15.79	6.40
0.05	1.48	6.01	32.11	37.13
0.10	0.57	9.71	2.64	32.78
0.20	0.09	1.67	0.18	3.61

Table 7.11: Average CPU times (in seconds) for different choices of Pr and n in BB2 and SSDP run on Ins^{TAN}.

in very short runtimes when there are no precedence constraints. SSDP performs worse, however, when the precedence graph is dense, while the B&B algorithms will tend to perform better exactly in this case. To conclude this comparison, we underline the fact that our algorithms have been developed to solve the more general setting in which time windows are also imposed, whereas the instance set examined here does not contain such time windows.

7.4.3.2 A single machine problem with time windows: $1|r_j, \delta_j| \sum w_j C_j$

Another special case of P is the single machine problem with time windows where the objective function is to minimize the total weighted sum of the completion times. We run our B&B algorithms on one of the instance sets provided by Pan and Shi (2005), which has been introduced as problem set (I) in which the parameters α and β define the ranges for the generation of release dates and deadlines, respectively. We refer to this instance set as Ins^{PAN}. To solve these instances, we set all due dates to zero. Table 7.12 shows the computational results of BB1 and BB2 applied to Ins^{PAN}. Our B&B algorithms both solve 394 out of the 400 instances to optimality within the time limit of 1200 seconds. Although a consistent pattern cannot be recognized, it seems that the hardest instances belong to the subsets where $\alpha = 1$ and $\beta = 16$. Since we do not have access to the code of Pan and Shi, direct comparisons are difficult, but overall our runtimes are of the same order of magnitude, although the most difficult instances for Pan and Shi are not the most difficult ones for our code, and vice versa.

Contrary to the discussion in Section 7.4.2, we notice that our two algorithms behave quite similarly for these instances. This can be explained as follows. First, for all members of Ins^{PAN}, release dates are non-zero, such that BB2 follows the same steps as BB1 until the release dates of all remaining jobs are less than the decision time. Second, the fact that due dates are zero makes all jobs late already in the root node and thus the scheduling of any job (even in the beginning of the schedule) has a positive contribution in the objective value. For the case where due dates are non-zero, scheduling backward is advantageous because jobs are mostly early in the beginning of the schedule, so they have zero contribution in the objective value.

7.5 Summary and conclusion

In this chapter, we have developed a branch-and-bound algorithm that solves the instances of the GSMSP to guaranteed optimality. Computational results show that our approach is effective in solving medium-sized instances, and that it compares favorably with two straightforward linear formulations. Our procedure was also compared with two existing methods, namely an SSDP algorithm and a B&B algorithm, for special cases of the problem. The SSDP algorithm requires only very low runtimes in the absence of precedence constraints, but it performs worse when the precedence graph is dense, which is exactly the easiest setting for our B&B algorithms.

n	α	β	Method	
			BB1	BB2
20	0.5	1	0.006	0.005
		2	0.008	0.005
		4	0.010	0.008
		8	0.012	0.010
		16	0.012	0.010
	1	1	0.002	0.002
		2	0.002	0.002
		4	0.004	0.004
		8	0.013	0.011
		16	0.009	0.008
30	0.5	1	0.026	0.022
		2	0.040	0.047
		4	0.055	0.053
		8	0.100	0.110
		16	0.105	0.107
	1	1	0.007	0.008
		2	0.033	0.033
		4	0.017	0.018
		8	0.161	0.160
		16	0.092	0.087
40	0.5	1	0.111	0.123
		2	0.306	0.301
		4	1.419	1.451
		8	2.512	2.477
		16	0.987	0.967
	1	1	0.014	0.012
		2	0.234	0.165
		4	0.161	0.170
		8	0.267	0.282
		16	0.867	0.875
50	0.5	1	0.782	0.784
		2	3.028	3.038
		4	11.082	11.152
		8	17.801	17.718
		16	100.041	100.253
	1	1	0.036	0.038
		2	1.660	1.638
		4	13.466(1)	13.686(1)
		8	71.697(2)	72.407(2)
		16	77.316(3)	77.406(3)

Table 7.12: Average CPU times (in seconds; first number) and number of unsolved instances within the time limit (between brackets, if any; out of 10) for different choices of n, α and β in BB1 and BB2 run on Ins^{PAN}.

List of Figures

List of Tables

Bibliography

Abbasi, B., Shadrokh, S., and Arkat, J. Bi-objective resource-constrained project scheduling with robustness and makespan criteria. *Applied mathematics and computation*, 180(1):146–152, 2006.

Abdul-Razaq, T. and Potts, C. Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society*, 39(2):141–152, 1988.

Abdul-Razaq, T., Potts, C., and Wassenhove, L. V. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26:235–253, 1990.

Akkan, C., Külünk, M. E., and Koçaş, C. C. Finding robust timetables for project presentations of student teams. *European Journal of Operational Research*, 249(2):560–576, 2016.

Akturk, M. and Ozdemir, D. An exact approach to minimizing total weighted tardiness with release dates. *IIE Transactions*, 32:1091–1101, 2000.

Akturk, M. and Ozdemir, D. A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational Research*, 135(2):394–412, 2001.

Al-Fawzan, M. A. and Haouari, M. A bi-objective model for robust resource-constrained project scheduling. *International Journal of Production Economics*, 96(2):175–187, 2005.

Alvarez-Valdes, R. and Tamarit, J. The project scheduling polyhedron: Dimension, facets and lifting theorems. *European Journal of Operational Research*, 67(2):204–220, 1993.

Artigues, C., Michelon, P., and Reusser, S. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249–267, 2003.

Artigues, C., Demassey, S., and Neron, E. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. Wiley, 2008.

Artigues, C., Leus, R., and Talla Nobibon, F. Robust optimization for resource-constrained project scheduling with uncertain activity durations. *Flexible Services and Manufacturing Journal*, 25(1-2):175–205, 2013.

Ashtiani, B., Leus, R., and Aryanezhad, M.-B. New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of pre-processing. *Journal of Scheduling*, 14(2): 157–171, 2011.

Aytug, H., Lawley, M., McKay, K., Mohan, S., and Uzsoy, R. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1): 86–110, 2005.

Baker, K. *Introduction to Sequencing and Scheduling*. Wiley, 1974.

Ballestín, F. and Leus, R. Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Production and Operations Management*, 18:459–474, 2009.

Ballestín, F. When is it worthwhile to work with the stochastic RCPSP? *Journal of Scheduling*, 10(3):153–166, 2007.

Ballestín, F. and Trautmann, N. An iterated-local-search heuristic for the resource-constrained weighted earliness-tardiness project scheduling problem. *International Journal of Production Research*, 46(22):6231–6249, 2008.

Bean, J. C., Birge, J. R., Mittenthal, J., and Noon, C. E. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, 1991.

Bein, W., Kamburowski, J., and Stallmann, M. Optimal reduction of two-terminal directed acyclic graphs. *SIAM Journal on Computing*, 21 (6):1112–1129, 1992.

Belouadah, H., Posner, M., and Potts, C. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics*, 36(3):213–231, 1992.

Ben-Tal, A., Goryashko, A., Guslitzer, E., and Nemirovski, A. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, 2003.

Birge, J. R. and Louveaux, F. *Introduction to Stochastic Programming*. Springer, 2011.

Blazewicz, J., Lenstra, J. K., and Rinnooy Kan, A. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

Briskorn, D., Leung, J., and Pinedo, M. Robust scheduling on a single machine using time buffers. *IIE Transactions*, 43(6):383–398, 2011.

Bruni, M. E., Beraldi, P., Guerriero, F., and Pinto, E. A heuristic approach for resource constrained project scheduling with uncertain activity durations. *Computers & Operations Research*, 38(9):1305–1318, 2011.

Calinescu, G., Fernandes, C., Kaul, H., and Zelikovsky, A. Maximum series-parallel subgraph. *Algorithmica*, 63(1-2):137–157, 2012.

Chaari, T., Chaabane, S., Aissani, N., and Trentesaux, D. Scheduling under uncertainty: Survey and research directions. In *Advanced Logistics and Transport (ICALT), 2014 International Conference on*, pages 229–234, 2014.

Chapman, C. and Ward, S. *Project Risk Management: Processes, Techniques and Insights*. John Wiley & Sons, 2007.

Christofides, N., Alvarez-Valdes, R., and Tamarit, J. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29:262–273, 1987.

Chtourou, H. and Haouari, M. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers & Industrial Engineering*, 55(1):183–194, 2008.

Cong, J. Computing maximum weighted k-families and k-cofamilies in partially ordered sets. Technical report, University of California, 1993.

Conway, R., Maxwell, W., and Miller, L. *Theory of Scheduling*. Addison Wesley, Reading, MA, 1967.

Creemers, S. Minimizing the expected makespan of a project with stochastic activity durations under resource constraints. *Journal of Scheduling*, 18(3):263–273, 2015.

Creemers, S., Leus, R., and Lambrecht, M. Scheduling markovian PERT networks to maximize the net present value. *Operations Research Letters*, 38(1):51–56, 2010.

Davari, M. and Demeulemeester, E. The proactive and reactive resource-constrained project scheduling problem. Technical Report KBI_1613, KU Leuven, 2016a.

Davari, M. and Demeulemeester, E. A novel branch-and-bound algorithm for the chance-constrained rcpsp. Technical Report KBI_1620, KU Leuven, 2016b.

Davari, M. and Demeulemeester, E. The proactive and reactive resource-constrained project scheduling problem: The crucial role of buffer-based reactions. Technical report, KU Leuven, 2017.

Davari, M., Demeulemeester, E., Leus, R., and Nobibon, F. T. Exact algorithms for single-machine scheduling with time windows and precedence constraints. *Journal of Scheduling*, 19:309–334, 2016.

De Reyck, B. and Leus, R. R&D project scheduling when activities may fail. *IIE transactions*, 40(4):367–384, 2008.

Debels, D. and Vanhoucke, M. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457–469, 2007.

Deblaere, F., Demeulemeester, E., and Herroelen, W. RESCON: Educational project scheduling software. *Computer Applications in Engineering Education*, 19(2):327–336, 2011a.

Deblaere, F., Demeulemeester, E., and Herroelen, W. Reactive scheduling in the multi-mode RCPSP. *Computers & Operations Research*, 38(1): 63–74, 2011b.

Deblaere, F., Demeulemeester, E., and Herroelen, W. Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, 214(2):308–316, 2011c.

Demassey, S., Artigues, C., and Michelon, P. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal on computing*, 17(1):52–65, 2005.

Demeulemeester, E. and Herroelen, W. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818, 1992.

Demeulemeester, E. and Herroelen, W. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43:1485–1492, 1997.

Demeulemeester, E., Vanhoucke, M., and Herroelen, W. RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6:17–38, 2003.

Demeulemeester, E. and Herroelen, W. *Project Scheduling: A Research Handbook*. Kluwer Academic Publisher, 2002.

Demeulemeester, E. and Herroelen, W. Robust project scheduling. *Foundations and Trends in Technology, Information and Operations Management*, 3:201–376, 2011.

Duenas, A. and Petrovic, D. An approach to predictive-reactive scheduling of parallel machines subject to disruptions. *Annals of Operations Research*, 159(1):65–82, 2008.

Dyer, M. and Wolsey, L. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(2–3):255–270, 1990.

Fisher, M. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

Flyvbjerg, B., Bruzelius, N., and Rothengatter, W. *Megaprojects and Risk: An Anatomy of Ambition*. Cambridge University Press, 2003.

Fu, N., Lau, H. C., Varakantham, P., and Xiao, F. Robust local search for solving RCPSP/max with durational uncertainty. *Journal of Artificial Intelligence Research*, 43(1):43–86, 2012.

Gabrel, V., Murat, C., and Thiele, A. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235 (3):471 – 483, 2014.

Garey, M. and Johnson, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

Goldratt, E. *Critical Chain*. The North River Press Publishing Corporation, Great Barrington, 1997.

Golumbic, M. C. CHAPTER 5 - comparability graphs. In Golumbic, M. C., editor, *Algorithmic Graph Theory and Perfect Graphs*, pages 105–148. Academic Press, 1980.

Gordon, V., Potapneva, E., and Werner, F. Single machine scheduling with deadlines, release and due dates. *Optimization*, 42(3):219–244, 1997.

Graham, R., Lawler, E., Lenstra, J., and Kan, A. R. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

Graham, R. L. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

Grötschel, M., Lovász, L., and Schrijver, A. Polynomial algorithms for perfect graphs. In Berge, C. and Chvátal, V., editors, *Topics on Perfect Graphs*, volume 88 of *North-Holland Mathematics Studies*, pages 325–356. North-Holland, 1984.

Hariri, A. and Potts, C. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics*, 5(1):99–109, 1983.

Hartmann, S. and Kolisch, R. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127:394–407, 2000.

Hartmann, S. and Briskorn, D. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

Held, M. and Karp, R. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

Herroelen, W. and Leus, R. On the merits and pitfalls of critical chain scheduling. *Journal of Operations Management*, 128(3):221–230, 2001.

Herroelen, W. and Leus, R. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165: 289–306, 2005.

Herroelen, W., Leus, R., and Demeulemeester, E. Critical chain project scheduling: Do not oversimplify. *Project Management Journal*, 33:48–60, 2002.

Herroelen, W. Project scheduling: Theory and practice. *Production and Operations Management*, 14(4):413–432, 2005.

Herroelen, W. Generating robust project baseline schedules. In *Tutorials in operations research*. INFORMS, 2007.

Herroelen, W. and Leus, R. The construction of stable project baseline schedules. *European Journal of Operational Research*, 156(3):550–565, 2004a.

Herroelen, W. and Leus, R. Robust and reactive project scheduling: A review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004b.

Hillson, D. Addressing risk. PM Network, October 2003.

Hoogeveen, J. and van de Velde, S. Stronger Lagrangian bounds by use of slack variables: Applications to machine scheduling problems. *Mathematical Programming*, 70:173–190, 1995.

Ibaraki, T. and Nakamura, Y. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76(1): 72–82, 1994.

Igelmund, G. and Radermacher, F. J. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13(1):1–28, 1983a.

Igelmund, G. and Radermacher, F. J. Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13(1): 29–48, 1983b.

Jouglet, A., Baptiste, P., and Carlier, J. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chapter 13, Branch and Bound Algorithms for Total Weighted Tardiness. CRC Press, Boca Raton, FL, USA, 2004.

Kaerkes, R. Netzplan theory. *Methods of Operations Research*, 27:1–65, 1977.

Kaerkes, R. and Leipholz, B. Generalized network functions in flow networks. *Methods of Operations Research*, 27:441–465, 1977.

Kaplan, L. A. *Resource-constrained project scheduling with preemption of jobs*. University of Michigan, 1988.

Keha, A., Khowala, K., and Fowler, J. Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, 56:357–367, 2009.

Kéri, A. and Kis, T. Primal-dual combined with constraint propagation for solving RCPSPWET. In *Operations Research Proceedings 2005*, pages 685–690. Springer, 2006.

Khemakhem, M. A. and Chtourou, H. Efficient robustness measures for the resource-constrained project scheduling problem. *International Journal of Industrial and Systems Engineering*, 14(2):245–267, 2013.

Kinable, J., Wauters, T., and Berghe, G. V. The concrete delivery problem. *Computers & Operations Research*, 48:53–68, 2014.

Klastorin, T. and Mitchell, G. Optimal project planning under the threat of a disruptive event. *IIE Transactions*, 45(1):68–80, 2013.

Klein, R. *Scheduling of Resource Constrained Projects*. Kluwer Academic Publisher, 2000.

Klimek, M. and Łebkowski, P. Robust buffer allocation for scheduling of a project with predefined milestones. *Decision Making in Manufacturing and Services*, 3(2):49, 2009.

Kolisch, R. and Hartmann, S. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, 2006.

Kolisch, R. and Sprecher, A. PSPLIB - a project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216, 1997.

Koné, O., Artigues, C., Lopez, P., and Mongeau, M. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3 – 13, 2011.

Küçükyavuz, S. On mixing sets arising in chance-constrained programming. *Mathematical Programming*, 132(1):31–56, 2012.

Kuster, J., Jannach, D., and Friedrich, G. Extending the RCPSP for modeling and solving disruption management problems. *Applied Intelligence*, 31(3):234–253, 2009.

Kuster, J., Jannach, D., and Friedrich, G. Applying local rescheduling in response to schedule disruptions. *Annals of Operations Research*, 180 (1):265–282, 2010.

Lamas, P. and Demeulemeester, E. A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *Journal of Scheduling*, 19:409–428, 2016.

Lambrechts, O., Demeulemeester, E., and Herroelen, W. Exact and suboptimal reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. Technical report, 2007.

Lambrechts, O., Demeulemeester, E., and Herroelen, W. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of Scheduling*, 11(2):121–136, 2008a.

Lambrechts, O., Demeulemeester, E., and Herroelen, W. A tabu search procedure for developing robust predictive project schedules. *International Journal of Production Economics*, 111(2):493–508, 2008b.

Lambrechts, O., Demeulemeester, E., and Herroelen, W. Time slack-based techniques for robust project scheduling subject to resource uncertainty. *Annals of Operations Research*, 186(1):443–464, 2011.

Lawler, E. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, 1973.

Lawler, E. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

Lawler, E. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Algorithmic Aspects of Combinatorics*, 2:75–90, 1978.

Lenstra, J., Kan, A. R., and Brucker, P. Complexity of machine scheduling problems. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.

Leus, R. *The generation of stable project plans*. PhD thesis, Department of applied economics, Katholieke Universiteit Leuven, Belgium, 2003.

Leus, R. and Herroelen, W. Stability and resource allocation in project planning. *IIE Transactions*, 36(7):1–16, 2004.

Leus, R. and Herroelen, W. The complexity of machine scheduling for stability with a single disrupted job. *Operations Research Letters*, 33: 151–156, 2005.

Leus, R. Resource allocation by means of project networks: Complexity results. *Networks*, 58(1):59–67, 2011a.

Leus, R. Resource allocation by means of project networks: Dominance results. *Networks*, 58(1):50–58, 2011b.

Liebchen, C., Lübbecke, M., Möhring, R., and Stiller, S. *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, chapter The Concept of Recoverable Robustness, Linear Programming Recovery, and Railway Applications, pages 1–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

Lock, D. *Project Management*. Gower, Nineth edition, 2007.

Luedtke, J. and Ahmed, S. A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 19(2):674–699, 2008.

Luedtke, J., Ahmed, S., and Nemhauser, G. L. An integer programming approach for linear programs with probabilistic constraints. *Mathematical Programming*, 122(2):247–272, 2010.

McMahon, G. and Lim, C. The two-machine flow shop problem with arbitrary precedence relations. *European Journal of Operational Research*, 64(2):249–257, 1993.

Mehta, S. V. Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12(1):15–38, 1999.

Mehta, S. and Uzsoy, R. Predictive scheduling of a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation*, 14: 365–378, 1998.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44(5):714–729, 1998.

Möhring, R., Radermacher, F., and Weiss, G. Stochastic scheduling problems I - Set strategies. *Mathematical Methods of Operations Research*, 28:193–260, 1984.

Möhring, R., Radermacher, F., and Weiss, G. Stochastic scheduling problems II - General strategies. *Mathematical Methods of Operations Research*, 29:65–104, 1985a.

Möhring, R., Schulz, A., Stork, F., and Uetz, M. Solving project scheduling problems by minimum cut computations. *Mathematical Methods of Operations Research*, 29:65–104, 2000.

Möhring, R., Radermacher, F., and Weiss, G. Stochastic scheduling problems II - set strategies. *Mathematical Methods of Operations Research*, 29(3):65–104, 1985b.

Nemeth, G., Lovrek, I., and Sinkovic, V. Scheduling problems in parallel systems for telecommunications. *Computing*, 58:199–223, 1997.

Nessah, R. and Kacem, I. Branch-and-bound method for minimizing the weighted completion time scheduling problem on a single machine with release dates. *Computers & Operations Research*, 39(3):471–478, 2012.

Neumann, K., Schwindt, C., and Zimmermann, J. *Project scheduling with time windows and scarce resources.* Springer, 2003.

Pan, Y. An improved branch and bound algorithm for single machine scheduling with deadlines to minimize total weighted completion time. *Operations Research Letters*, 31(6):492–496, 2003.

Pan, Y. and Shi, L. Dual constrained single machine sequencing to minimize total weighted completion time. *IEEE Transactions on Automation Science and Engineering*, 2(4):344–357, 2005.

Pinedo, M. *Scheduling: Theory, Algorithms, and Systems.* Springer, 2008.

Pinto, J. K. and Prescott, J. E. Planning and tactical factors in the project implementation process. *Journal of Management Studies*, 27 (3):305–327, 1990.

Posner, M. Minimizing weighted completion times with deadlines. *Operations Research*, 33:562–574, 1985.

Potts, C. A lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Science*, 31(10):1300–1311, 1985.

Potts, C. and Van Wassenhove, L. An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *European Journal of Operational Research*, 12(4):379–387, 1983.

Potts, C. and Van Wassenhove, L. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2):363–377, 1985.

Prékopa, A. Dual method for the solution of a one-stage stochastic programming problem with random rhs obeying a discrete probability distribution. *Mathematical Methods of Operations Research*, 34(6):441–461, 1990.

Pritsker, A., Watters, L., and Wolfe, P. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108, 1969.

Radermacher, F. Cost-dependent essential systems of es-strategies for stochastic scheduling problems. *Methods of Operations Research*, 42: 17–31, 1981.

Radermacher, F. *Kapazitätsoptimierung in Netzplänen.* Mathematical systems in economics. Oelgeschlager, Gunn & Hain, Cambridge (MA), 1978.

Rostami, S., Creemers, S., and Leus, R. New benchmark results for the stochastic resource-constrained project scheduling problem. *Journal of Scheduling*, to appear, 2017.

Ruszczyński, A. Probabilistic programming with discrete distributions and precedence constrained knapsack polyhedra. *Mathematical Programming*, 93(2):195–215, 2002.

Sabuncuoglu, I. and Goren, S. Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research. *International Journal of Computer Integrated Manufacturing*, 22(2):138–157, 2009.

Schwindt, C. *Resource Allocation in Project Management.* Springer, 2005.

Schwindt, C. and Zimmermann, J., editors. *Handbook on Project Management and Scheduling Vol. 1.* International Handbooks on Information Systems. Springer International Publishing, 2015a.

Schwindt, C. and Zimmermann, J., editors. *Handbook on Project Management and Scheduling Vol. 2.* International Handbooks on Information Systems. Springer International Publishing, 2015b.

Shapiro, A. A dynamic programming approach to adjustable robust optimization. *Operations Research Letters*, 39(2):83 – 87, 2011.

Shirazi, B., Kavi, K., and Hurson, A., editors. *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Society Press, 1995.

Sprecher, A. Scheduling resource-constrained projects competitively at modest resource requirements. *Management Science*, 46:710–723, 2000.

Stork, F. *Stochastic resource-constrained project scheduling.* PhD thesis, Technical University of Berlin, School of Mathematics and Natural Sciences, 2001.

Stork, F. Branch-and-bound algorithms for stochastic resource-constrained project scheduling. Technical report, 2000.

Sule, D. *Production Planning and Industrial Scheduling: Examples, Case Studies and Applications*. CRC Press, 2007.

Tahooneh, A. and Ziarati, K. Using artificial bee colony to solve stochastic resource constrained project scheduling problem. In *Advances in Swarm Intelligence*, pages 293–302. Springer, 2011.

Talla Nobibon, F. and Leus, R. Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research*, 38(1):367–378, 2011.

Tanaka, S. and Fujikuma, S. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. *Journal of Scheduling*, 15:347–361, 2012.

Tanaka, S. and Sato, S. An exact algorithm for the precedence-constrained single-machine scheduling problem. *European Journal of Operational Research*, 229(2):345–352, 2013.

Tanaka, S., Fujikuma, S., and Araki, M. An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling*, 12:575–593, 2009.

Tang, L., Xuan, H., and Liu, J. Hybrid backward and forward dynamic programming based Lagrangian relaxation for single machine scheduling. *Computers & Operations Research*, 34(9):2625–2636, 2007.

Valdes, J., Tarjan, R., and Lawler, E. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982.

van de Velde, S. Dual decomposition of a single-machine scheduling problem. *Mathematical Programming*, 69(1-3):413–428, 1995.

Van de Vonder, S. *Proactive-reactive procedures for robust project scheduling*. PhD thesis, Department of Decision Sciences and Information Management (KBI), K.U. Leuven, 2006.

Van de Vonder, S., Demeulemeester, E., Herroelen, W., and Leus, R. The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, 97: 227–240, 2005.

Van de Vonder, S., Demeulemeester, E., Leus, R., and Herroelen, W. *Proactive/reactive project scheduling - Trade-offs and procedures*, pages 25–51. 2006. in Jozefowska J, Weglarz J, ed.: Perspectives in Modern Project Scheduling, (2).

Van de Vonder, S., Ballestín, F., Demeulemeester, E., and Herroelen, W. Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering*, 52(1):11–28, 2007a.

Van de Vonder, S., Demeulemeester, E., and Herroelen, W. A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, 10(3):195–207, 2007b.

Van de Vonder, S., Demeulemeester, E., and Herroelen, W. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189(3):723–733, 2008.

Van den Akker, J., Diepen, G., and Hoogeveen, J. Minimizing total weighted tardiness on a single machine with release dates and equal-length jobs. *Journal of Scheduling*, 13:561–576, 2010.

Vanhoucke, M., Demeulemeester, E., and Herroelen, W. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*, 102:179–196, 2001.

Xu, J. and Parnas, D. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transaction on Software Engineering*, 16(3):360–369, 1990.

Zheng, Z., Shumin, L., Ze, G., and Yueni, Z. Resource-constrained multi-project scheduling with priorities and uncertain activity durations. *International Journal of Computational Intelligence Systems*, 6(3):530–547, 2013.

Zhu, G., Bard, J., and Yu, G. Disruption management for resource-constrained project scheduling. *Journal of the Operational Research Society*, 56:365–381, 2005.

Zhu, G., Bard, J. F., and Yu, G. A two-stage stochastic programming approach for project planning with uncertain activity durations. *Journal of Scheduling*, 10(3):167–180, 2007.

Doctoral Dissertations from the Faculty of Business and Economics

A list of doctoral dissertations from the Faculty of Business and Economics can be found at the following website:
http://www.econ.kuleuven.be/phd/doclijst.htm.